Oracle® Insurance IStream

IStream Communicator Technical Guide

Release 2.2 FP3

E14918-01

April 2008

ORACLE

INSURANCE

Copyright

Primary Authors: Andrew Brooke and Ken Weinberg

CONTENTS

Chapter 1

Introduction

Note: In this release, *InSystems Correspondence* has been renamed to *IStream Communicator.*

Welcome to the IStream Communicator Technical Guide.

This guide provides information, samples, and instruction on using the Integrated Data Mapper (IDM) and the Communicator Software Developers Kit (SDK). It is for technical users such as system administrators and developers.

This guide describes how to:

- prepare custom coding to retrieve data for Communicator
- integrate your applications with Communicator
- prepare Calligo Model Documents for Communicator
- configure `CorrConfig.xml`

The chapter contains the following topics:

Document Conventions

Tips, Notes, Important Notes and Warnings

Tip: A **Tip** provides a better way to use the software.

Note: A **Note** contains special information and reminders.

Important: An **Important** note contains significant information about the use and understanding of the software.

Warning: A **Warning** contains critical information that if ignored, may cause errors or result in the loss of information.

Other Document Conventions

- Product names are in italics, for example, *Microsoft Word*.
- Window names, buttons, tabs and other screen entities are in bold, for example: Click **Next**.
- Paths, URLs and code samples use the Courier font, for example: `C:\Windows`
- Some sections contain links to other **Related Topics**.

Overview of IStream Communicator

IStream Communicator is a J2EE enterprise solution that automates the process of creation, distribution and management of personalized ad-hoc correspondence.

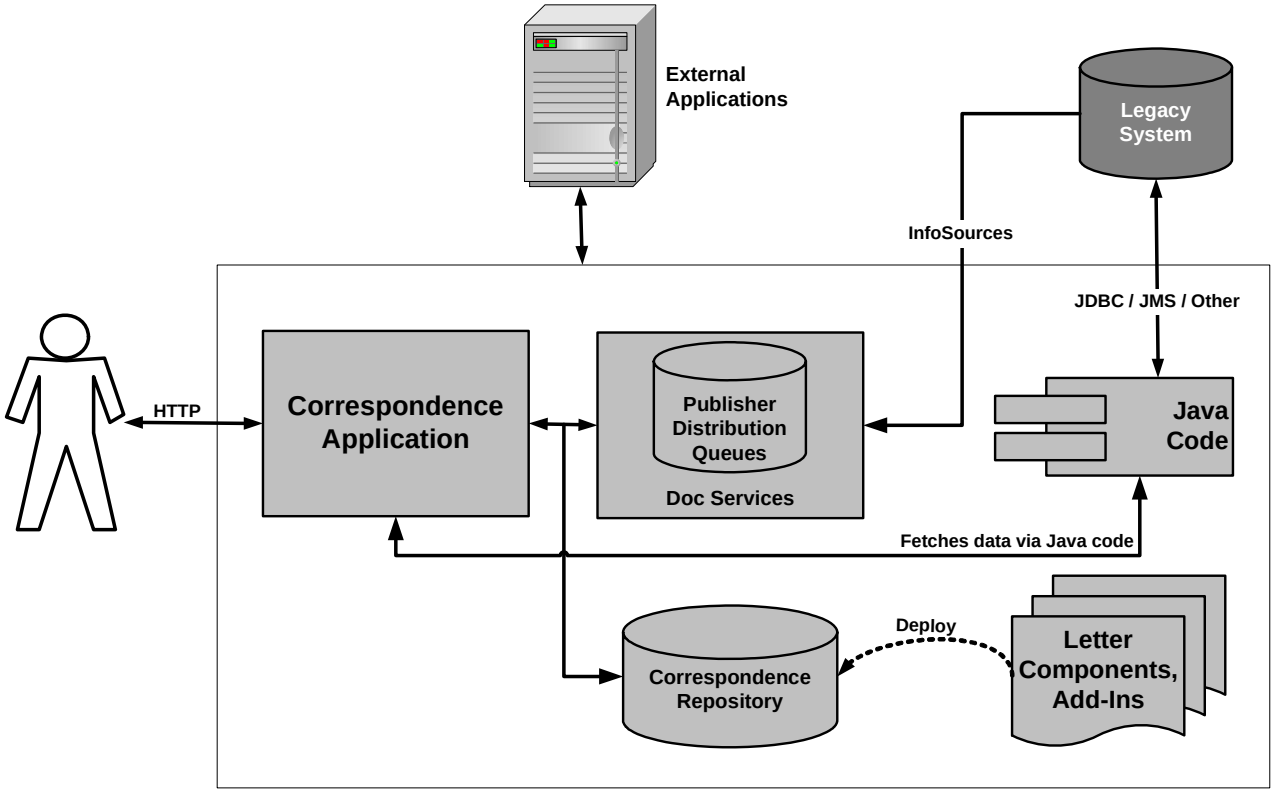
Communicator has a web-based user interface, and supports multiple distribution channels, such as Print, Fax and E-mail.

The Communicator architecture supports integration with legacy data using the Integrated Data Mapper.

Third party applications can integrate with Communicator using the Software Developer's Kit (SDK).

Application Architecture

The following diagram shows how the Communicator components relate to each other:



IStream Communicator Documentation

IStream Communicator includes the following documents on the installation CD. If you need a copy of any of these documents, please contact your system or product administrator.

- The *IStream Communicator User Guide* describes how each of the four roles (User, Contributor, Reviewer, and Administrator) can use Communicator. It contains overviews, step-by-step procedures and descriptions of the screens and fields.
- The *IStream Communicator Technical Guide* describes how to configure `CorrConfig.xml` using the Software Developer's Kit, and how to work with the integrated data mapper.
- The *IStream Communicator Installation Guide* includes system requirements and detailed installation and configuration information. It also includes requirements for installing Communicator with IStream Publisher or Calligo Enterprise.
- The *IStream Communicator Release Notes* contain general product information, product enhancements and new features, supported platforms and third-party software, and known limitations.
- The *IStream Communicator Online Help* contains the same information as the User's Guide, but in an online help format with a search tool, an index and a table of contents.

Contacting InSystems for Help

Customer Support hours are 8:00 A.M. to 8:00 P.M. (Eastern Standard Time), Monday through Friday. Outside of these hours, send us a detailed e-mail message and you will be contacted during regular business hours. Please provide detailed information, as described in the *Support Checklist*.

Contact Information

Mail:	Customer Support InSystems Corporation 19 Allstate Parkway, Suite 400 Markham, Ontario, L3R 5A4
Phone:	(905) 513-7466
Fax:	(905) 513-1684
Email:	support@insystems.com
Web:	www.insystems.com

Support Checklist

When contacting Customer Support, please provide the following information:

- Your name, company name, e-mail address, and phone number.
- Version numbers of all your InSystems products.
- Name and version of the network software.
- Windows version, including any installed Service Packs.
- .NET Framework version
- DMS version, including any installed Service Packs (if applicable).
- Microsoft Word version (if applicable).
- Database vendor and version (if applicable).
- Error messages and the circumstances of their occurrence.
- A full description of the problem:
    - What happened? What were the sequence of events that preceded the problem?
    - In which screen or window did the problem occur?
    - Was the problem the result of pressing a key?
    - Did the screen freeze? What functions of the software are affected?
    - How many people are affected?

Chapter 2

The Integrated Data Mapper

This chapter includes information about:

Note: *Integrated Data Mapper, Data Dictionary* and *Data Library* are interchangeable terms.

Generated Letter Data

This section explains how data arrives in a generated letter. It describes the hidden processes that occur before a Letter Definition can be created and before a letter can be generated.

In Communicator, people who create letter definitions and generate letters need to know that:

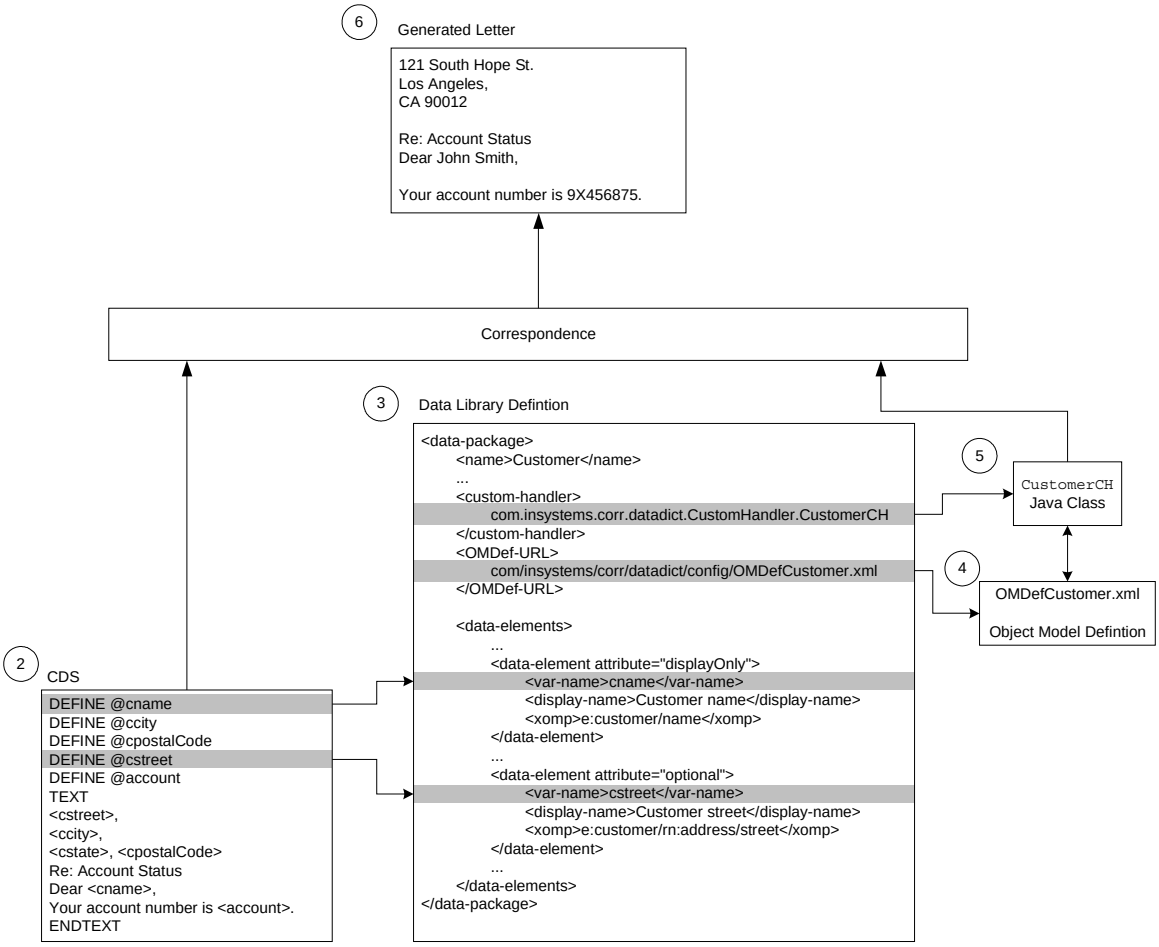
- a Communicator component definition version references a Calligo model document section (a .CDS file)
- at the end of the process, Communicator includes data from existing databases or sources in the generated letter.

Overview

The following steps give an overview of how data arrives in a generated letter:

1. Each Communicator Component Definition reference a Calligo Subsection (CDS file).
2. Each CDS file defines variables as part of the letter, for example the Customer address.
3. The Data Library Definition (an XML file) contains the CDS file's variables in a data package, and specifies which Object Model Definition and Data Package Custom Handler to use.
4. Each Object Model Definition specifies the structure and type of data in your existing application to the Data Package Custom Handler. The Object Model Definition is an XML representation of the data schema in your existing application.
5. Each Data Package Custom Handler uses the Object Model Definition to retrieve data from your existing data source, and return it as a data package that Communicator uses to generate a letter.
6. Communicator includes the data in the generated letter.

The following diagram illustrates this process:



Integrated Data Mapper Tutorial

This section provides a tutorial on how to create a Data Package for the Communicator Integrated Data Mapper (IDM). The tutorial is intended for technical professionals whose responsibilities include developing Data Packages and Data Package Custom Handlers for the Communicator IDM.

This tutorial provides information for the following tasks:

- creating an Object Model Definition XML from a Database Schema
- creating a Data Package for a Communicator Data Library
- developing a Data Package Custom Handler using the Java programming language

Full source code and XML files are in the `TowneInsuranceDPCH.jar` file supplied with Communicator.

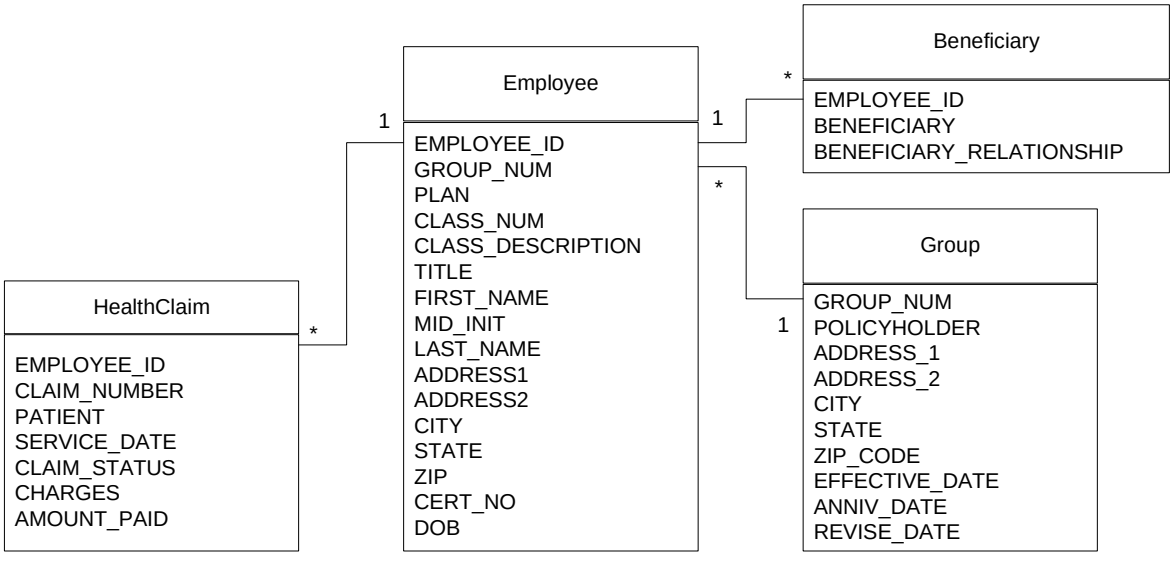
Towne Insurance

For our examples, we will use a fictional company, Towne Insurance, and specifically its Health Group.

Towne Insurance Health Group division uses a relational database to store information about its groups, employees, beneficiaries and claims. The information is stored in the Group database. This tutorial shows how to enable the Communicator application to access the information stored in the Group database.

Group Database Schema

There are four tables in the Group database. The following diagram describes the tables, columns, and relationships between the tables:



Group Object Model Definition

This section describes how to create an Object Model Definition from an existing database schema.

Note that:

- each table is represented by an entity in the model
- each column is represented by an entity property
- each relation is represented by an entity reference

The following code is in the `Group Object Model Definition - OMDefGroup.xml` file. The Object Model Definition for the Group database is in this file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE om SYSTEM "omdef.dtd">
<om id="GroupModel" schemaVersion="1.1">

   <entity id="e:Employee" independent="true">
      <property name="EmployeeID" type="java.lang.Integer"/>
      <property name="PlanName" type="java.lang.String"/>
      <property name="ClassNumber" type="java.lang.String"/>
      <property name="ClassDescription" type="java.lang.String"/>
      <property name="Title" type="java.lang.String"/>
      <property name="FirstName" type="java.lang.String"/>
      <property name="MiddleInitial" type="java.lang.String"/>
      <property name="LastName" type="java.lang.String"/>
      <property name="AddressLine1" type="java.lang.String"/>
      <property name="AddressLine2" type="java.lang.String"/>
      <property name="City" type="java.lang.String"/>
      <property name="State" type="java.lang.String"/>
      <property name="ZipCode" type="java.lang.String"/>
      <property name="CertificateNumber" type="java.lang.String"/>
      <property name="DateOfBirth" type="java.util.Date"/>
      <entityRef entityId="e:Group" roleName="rn:Group"
         cardinalityLowerBound="1" cardinalityUpperBound="1"/>
      <entityRef entityId="e:Beneficiary" roleName="rn:Beneficiary"
         cardinalityLowerBound="0" cardinalityUpperBound="-1"/>
      <entityRef entityId="e:HealthClaim" roleName="rn:HealthClaim"
         cardinalityLowerBound="0" cardinalityUpperBound="-1"/>
   </entity>

   <entity id="e:Group" independent="false">
      <property name="GroupName" type="java.lang.String"/>
      <property name="AddressLine1" type="java.lang.String"/>
      <property name="AddressLine2" type="java.lang.String"/>
      <property name="City" type="java.lang.String"/>
      <property name="State" type="java.lang.String"/>
      <property name="ZipCode" type="java.lang.String"/>
      <property name="EffectiveDate" type="java.util.Date"/>
      <property name="AnniversaryDate" type="java.util.Date"/>
      <property name="RevisionDate" type="java.util.Date"/>
   </entity>
```

```
   <entity id="e:Beneficiary" independent="false">
      <property name="BeneficiaryName" type="java.lang.String" />
      <property name="RelationshipToSelf" type="java.lang.String"/>
   </entity>

   <entity id="e:HealthClaim" independent="false">
      <property name="PatientName" type="java.lang.String"/>
      <property name="ServiceDate" type="java.util.Date"/>
      <property name="ClaimStatus" type="java.lang.String"/>
      <property name="Charges" type="java.lang.Integer"/>
      <property name="AmountPaid" type="java.lang.Integer"/>
      <property name="ClaimNumber" type="java.lang.String"/>
   </entity>

</om>
```

Entity Reference

An entity reference represents a relation between two entities; in this case, a relation between tables in the database. Each reference is defined by `<entityRef>` tag and has four attributes:

- `entityId` – the ID of the referenced entity
- `roleName` – the name of the reference
- `cardinalityLowerBound` – the minimum number of related entities
- `cardinalityUpperBound` – the maximum number of related entities

Object Model Definition file

In the following example, the reference states that an `Employee` entity can have exactly one related `Group` entity. A `Group` includes many `Employees` but from an `Employee` perspective, an `Employee` belongs to only one *Group*.

```
<entityRef entityId="e:Group" roleName="rn:Group"
   cardinalityLowerBound="1"
   cardinalityUpperBound="1"/>
```

However in this next example, an `Employee` entity can have no `HealthClaims` or can have an unlimited number.

```
<entityRef entityId="e:HealthClaim"
roleName="rn:HealthClaim"
   cardinalityLowerBound="0"
   cardinalityUpperBound="-1"/>
```

Independent Attribute

Note the `independent` attribute for the `<entity>` tag. When there are no references to the particular entity, then this entity is independent and the attribute must be set to `true`.

For detailed information on the Object Model Definition file format, see *Object Model Definitions* on page 39.

Communicator Data Package

The Communicator Data Package is a view on a specific Object Model Definition. It defines business variables that are accessible from Calligo sections. The Data Package also provides Communicator with the information about the appearance of these variables within the Communicator user interface.

The Data Package definition is part of the Data Library Definition file. (A complete discussion of this file is beyond the scope of this tutorial.) Listed below are the key fragments of the Group Data Package Definition. A full listing of the Group Data Package Definition is in the `DataDict.xml` file.

Data Package Definition

```
<data-package>
   <name>Group</name>
   <description> This is Group data package </description>

   <custom-handler>
      com.towne.insurance.correspondence.GroupDPCH
   </custom-handler>

   <OMDef-URL>
      /com/towne/insurance/correspondence/OMDefGroup.xml
   </OMDef-URL>

   <data-elements>
      <data-element attribute="isKey">
         <var-name>Employee_ID</var-name>
         <display-name>Employee ID</display-name>
         <xomp>e:Employee/EmployeeID</xomp>
         <note>Enter Employee ID</note>
      </data-element>
      ...
      <data-element attribute="required">
         <var-name>Employee_FirstName</var-name>
         <display-name>First Name</display-name>
         <xomp>e:Employee/FirstName</xomp>
         <note>Employee first name</note>
      </data-element>
      ...
      <data-element attribute="required">
         <var-name>Patient_Name</var-name>
         <display-name>Claim Patient Name</display-name>
         <xomp>e:Employee/rn:HealthClaim/PatientName</xomp>
      </data-element>
      <data-element attribute="required">
         <var-name>Claim_Number</var-name>
         <display-name>Claim Number</display-name>
         <xomp>e:Employee/rn:HealthClaim/ClaimNumber</xomp>
      </data-element>
```

```
      ...
               <data-element attribute="required">
         <var-name>Beneficiary_Name</var-name>
         <display-name>Beneficiary Name</display-name>
         <xomp>e:Employee/rn:Beneficiary/BeneficiaryName</xomp>
      </data-element>
      ...
      <data-element attribute="required">
         <var-name>Group_Name</var-name>
         <display-name>Group Name</display-name>
         <xomp>e:Employee/rn:Group/GroupName</xomp>
      </data-element>
      ...
   </data-elements>
</data-package>
```

<custom-handler> tag

The text in the body of this tag specifies the fully qualified Java class name of the Data Package Custom Handler. This class contains the Java code that retrieves the data values from the database.

<OMDef-URL> tag

The text in the body of this tag specifies the full path to the Object Model Definition file for this data package.

<data-element> tag

This tag defines a business variable that is accessible from Calligo sections. Sub tags of this tag define various information about the variable, such as its name, presentation, and where to find the value.

The variable name that Calligo sections use is in the `<var-name>` tag.

The name that users will see on the screen while creating the letter is in the `<display-name>` tag.

The location of the variable value in the Object Model is in the `<xomp>` tag.

Examples of the `<xomp>` values for data elements are:

```
<var-name>Employee_FirstName</var-name>
<xomp>e:Employee/FirstName</xomp>
```

The value for the variable `Employee_FirstName` is in the `FirstName` property of the `Employee` entity.

```
<var-name>Patient_Name</var-name>
<xomp>e:Employee/rn:HealthClaim/PatientName</xomp>
```

The value for the variable `Patient_Name` is in the `PatientName` property of the entity related to `Employee` entity by the `HealthClaim` role. In this case, it is the `HealthClaim` entity.

For detailed information on Data Packages and the Data Library Definition file, see *Data Dictionary Definition* on page 34.

Data Package Custom Handler

The Data Package Custom Handler is a Java class that retrieves data values from the database, or any other data source, and presents these values in a format that the Communicator Data Library can use.

This class will:

1. Construct entities and relations between entities according to the Object Model Definition.
2. Populate the entities with the data from the data source.
3. Indicate error conditions to the Data Library using correct exceptions.

When Communicator requests data from a particular data package, the Data Library determines the appropriate Data Package Custom Handler, initializes it and requests the data from it. After successfully retrieving the data values, the Data Library reads the values from the Object Model entities returned by the custom handler and forwards them to Communicator.

For a complete reference of the Data Package Custom Handler, see *Data Package Custom Handler Programmer's Reference* on page 43.

Towne Insurance Group Data Package Custom Handler

To show how the Data Package Custom Handler works, the following sample illustrates an implementation of this handler for the Group Object Model, and the data package used in the document.

The entire source code is in the `GroupDPCH.java` file in the `TowneInsuranceDPCH.jar`.

Data Package Custom Handler Header

```
/**
 * Sample Data Package Custom Handler for Towne Insurance Group Data
Package.
 *
 */
public class GroupDPCH implements DataPackageCustomHandler
{
```

In this example, the custom handler is implementing the `DataPackageCustomHandler` interface. The Communicator Data Library calls methods defined in the interface to interact with the custom handler.

init() Method

```
    /**
     * Initializes custom handler.
     *
     * @see DataPackageCustomHandler.init()
     *
     */
    public void init(Properties configuration, Logger logger)
        throws InitException
    {
        this.logger = logger;

        logger.debug(">>>GroupDPCH.init() enters");

        // Validate configuration parameters.
        // If some of the parameters are incorrect
        // indicate it to the Data Library by throwing the
        // InitException exception with the appropriate
        // message.
        //
        // ... validate the DataSource name
        //
        String dataSourceName = (String)
configuration.get("dataSourceName");

        if ((null == dataSourceName) || dataSourceName.equals(""))
        {
            throw new InitException("DataSource name is empty." +
                                    " Please check the configuration.");
        }

        // Connect to the database and throw InitException
        // if something is wrong in the process.
        //
        try
        {
            InitialContext ctx = new InitialContext();

            dataSource = (DataSource) ctx.lookup(dataSourceName);
        }
        catch (NamingException e)
        {
            logger.logException(e);

            String msg = "Cannot locate Datasource: " + dataSourceName;
            throw new InitException(msg);
        }

        logger.debug(">>>GroupDPCH.init() exits");
    }
```

The Communicator Data Mapper calls the `init()` method to initialize the custom handler. This method is an appropriate area to read the configuration, check the data source connection, and perform other initialization tasks.

release() Method

```
/**
 * Releases resources acquired in <code>init()</code>.
 *
 * @see DataPackageCustomHandler.release()
 *
 */
  public void release()
 {
     //
     // Release any resources acquired in the init() method
     //
 }
```

The Communicator Data Mapper calls the `release()` method to release the custom handler. This method is an appropriate area to release all acquired resources. In the case of the Group custom handler, there is nothing to release because this example uses a database connection pool.

getData() Method - Key Data Verification

Communicator Data Mapper calls the `getData()` method to retrieve data values.

```
/**
 * Retrieves the data from the Health Group database.
 *
 * @see DataPackageCustomHandler.getData()
 *
 */
public Map getData(Map keyData, ObjectModel omDef)
    throws DataException
{
    logger.debug(">>>GroupDPCH.getData() enters");

    // Retrieve and validate the key data values.
    // If some of the values are missing or of incorrect type
    // indicate it to the Data Library by throwing the
    // InvalidKeyDataException exception with the appropriate
    // message.
    //
    Object objKeyValue = keyData.get("Employee_ID");

    if ((null == objKeyValue))
    {
        throw new InvalidKeyDataException(
              "Missing [Employee_ID] key data value");
    }

    if (!(objKeyValue instanceof Integer))
    {
        String errMsg =
            "Incorrect [Employee_ID] key data type.\n" +
            "Expected java.lang.Integer, got " +
            objKeyValue.getClass().getName();
```

```
            logger.error(errMsg);
            throw new InvalidKeyDataException(errMsg);
        }
        Integer employeeId = (Integer) objKeyValue;
```

This method checks the key data value's presence and type. The `InvalidKeyDataException` exception is thrown if there is invalid key data.

getData() Method - Database Connection

```
        // Retrieve data from database
        //
        Connection con = null;

        try
        {
            // get the database connection from the pool
            //
            con = dataSource.getConnection();
```

Because this example uses a database connection pool, it obtains the connection just before it is uses it and then immediately releases when it is finished with it. This manages the connections more effectively

getData() Method - Construct and Populate Entities

```
            // Construct employee entity which is root entity for this
            // model
            //
            Entity employeeEntity = new Entity(omDef.getId(),
"e:Employee");
            String groupNum =
                populateEmployeeEntity(employeeEntity, employeeId, con);

            // Add related group entity
            //
            Entity groupEntity = new Entity(omDef.getId(), "e:Group");
            populateGroupEntity(groupEntity, groupNum, con);

            employeeEntity.addRelatedEntity("rn:Group", groupEntity);

            // Read beneficiaries from the database and add as related
            // entities to employee entity
            //
            Collection beneficiaries = getBeneficiaries(employeeEntity,
con);
            addRelatedEntities(employeeEntity,
                               "rn:Beneficiary", beneficiaries);

            // Read claims from the database and add as related
            // entities to employee entity
            //
            Collection claims = getClaims(employeeEntity, con);
```

```
            addRelatedEntities(employeeEntity, "rn:HealthClaim",
claims);

            // Constuct the resulting Map and add the root entity
            //
            Map entities = new HashMap();

            entities.put(employeeEntity.getEntityDefinitionId(),
                         employeeEntity);

            logger.debug("<<<GroupDPCH.getData() exits");

            return entities;
        }

        catch (InvalidClassTypeException e)
        {
            logger.logException(e);

            String errMsg =
                "An error occurred while accessing object model
definition.";
            throw new DataStructureException(errMsg);
        }
        catch (ObjectModelException e)
        {
            logger.logException(e);

            String errMsg =
                "An error occured while accessing object model
defintion.";
            throw new DataStructureException(errMsg);
        }
        catch (SQLException e)
        {
            logger.logException(e);

            String errMsg = "An error occured while accessing the
database.";
            throw new DataSourceException(errMsg);
        }
```

This will throw an appropriate Custom Handler exception if an error or exception occurs. For detailed information on Custom Handler exceptions, see *Data Package Custom Handler Programmer's Reference* on page 43.

getData() Method

```
        finally
        {
            try
            {
                con.close();
            }
            catch (SQLException e)
            {
                // nothing can be done if close() fails.
                //
                logger.warn(
                     "An error occured while closing SQL connection.");
                logger.logException(e);
            }
        }
    }
```

After the all data is retrieved, you need to release the database connection.

Helper Methods

populateEmployeeEntity() Method

The `populateEmployeeEntity()` helper method reads data from the database and populates the employee entity. In this sense, it "holds together" the database schema and the Object Model Definition.

```
    /**
     * Populates Employee entity with the data from database.
     *
     * @param employee Employee entity to populate with data.
     *
     * @param employeeId ID of the employee record in the database.
     *
     * @param con Database connection.
     *
     * @return A group number to which this employee belongs.
     *
     * @exception NoDataFoundException If employee record not found in
the
     *                                  database.
     *
     * @exception SQLException If an error occurred while executing SQL
     *            command.
     *
     * @exception ObjectModelException An error occurred while accessing
     *              object model definition.
     *
     * @exception InvalidClassTypeException An error occurred while
accessing
     *                                      object model definition.
     */
    private String populateEmployeeEntity(Entity employee,
                                          Integer employeeId,
```

```
                                            Connection con)
        throws NoDataFoundException, SQLException, ObjectModelException,
                    InvalidClassTypeException
    {
        // prepare and execute SQL statement
        //
        String sql = "SELECT * FROM HEALTH_EMPLOYEE where EMPLOYEE_ID =
?";

        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setInt(1, employeeId.intValue());

        ResultSet rs = stmt.executeQuery();

        // there is no employee record for the specified employeeId
        //
        if (!rs.next())
        {
            String errMsg =
                "No employee record found for employeeId=" + employeeId;
            logger.info(errMsg);

            throw new NoDataFoundException(errMsg);
        }

        // populate the employee entity
        //
        employee.setPropertyValue("EmployeeID", employeeId);
        employee.setPropertyValue("PlanName", rs.getString("plan"));
        employee.setPropertyValue("ClassNumber",
rs.getString("class_num"));
        employee.setPropertyValue("Title", rs.getString("title"));
        employee.setPropertyValue("FirstName",
rs.getString("first_name"));
        employee.setPropertyValue("MiddleInitial",
rs.getString("mid_init"));
        employee.setPropertyValue("LastName",
rs.getString("last_name"));
        employee.setPropertyValue("AddressLine1",
rs.getString("address1"));
        employee.setPropertyValue("AddressLine2",
rs.getString("address2"));
        employee.setPropertyValue("City", rs.getString("city"));
        employee.setPropertyValue("State", rs.getString("state"));
        employee.setPropertyValue("ZipCode", rs.getString("zip"));
        employee.setPropertyValue("DateOfBirth", rs.getDate("dob"));
        employee.setPropertyValue("ClassDescription",
                                  rs.getString("class_description"));
        employee.setPropertyValue("CertificateNumber",
                                  rs.getString("cert_no"));

        // group number is used as foreign key in other tables
        //
        return rs.getString("group_num");
```

```
    }
```

These methods are similar to the `populateEmployeeEntity()` helper. They read data from the database and populate the corresponding entities.

```
    /**
     * Populates Group entity with the data from database.
     *
     * @param group Group entity to populate with data.
     *
     * @param groupNum Group number (primary key) in the database.
     *
     * @param con Database connection.
     *
     *
     * @exception DataStructureException If group record not found in
the
     *                                     database.
     *
     * @exception SQLException If an error occurred while executing SQL
     *                         command.
     *
     * @exception ObjectModelException An error occurred while accessing
     *                                 object model definition.
     *
     * @exception InvalidClassTypeException An error occurred while
accessing
     *                                      object model definition.
     */
    private void populateGroupEntity(Entity group, String groupNum,
                                     Connection con)
        throws DataStructureException, SQLException,
ObjectModelException,
                   InvalidClassTypeException
    {
        // prepare and execute SQL statement
        //
        String sql = "SELECT * FROM HEALTH_GROUP where GROUP_NUM = ?";

        PreparedStatement stmt = con.prepareStatement(sql);
        stmt.setString(1, groupNum);

        ResultSet rs = stmt.executeQuery();

        // there is no group record for the specified groupNum
        // an employee must belong to a group, if group record not
        // found it is a violation of data integrity.
        //
        if (!rs.next())
        {
            String errMsg = "No group record found for groupNum=" +
groupNum;
            logger.error(errMsg);

            throw new DataStructureException(errMsg);
```

```
        }

        // populate the group entity
        //
        group.setPropertyValue("GroupName",
rs.getString("policyholder"));
        group.setPropertyValue("AddressLine1",
rs.getString("address_1"));
        group.setPropertyValue("AddressLine2",
rs.getString("address_2"));
        group.setPropertyValue("City", rs.getString("city"));
        group.setPropertyValue("State", rs.getString("state"));
        group.setPropertyValue("ZipCode", rs.getString("zip_code"));
        group.setPropertyValue("EffectiveDate",
                                    rs.getDate("effective_date"));
        group.setPropertyValue("AnniversaryDate",
rs.getDate("anniv_date"));
        group.setPropertyValue("RevisionDate",
rs.getDate("revise_date"));
    }



    /**
     * Retrieves beneficiaries from the database.
     *
     * @param employee Employee entity for which to retrieve the
     *                 beneficiaries.
     *
     * @param con Database connection.
     *
     * @return A collection of beneficiary entities for the employee.
     *
     * @exception SQLException If an error occurred while executing SQL
     *            command.
     *
     * @exception ObjectModelException An error occurred while accessing
     *              object model definition.
     *
     * @exception InvalidClassTypeException An error occurred while
accessing
     *                                      object model definition.
     */
    private Collection getBeneficiaries(Entity employee, Connection con)
        throws SQLException, ObjectModelException,
InvalidClassTypeException
    {
        Collection beneficiaries = new ArrayList();

        // Get the information from the database
        //
        String sql =
                "SELECT * FROM HEALTH_BENEFICIARY where EMPLOYEE_ID =
?";
```

```
        PreparedStatement stmt = con.prepareStatement(sql);

        Integer employeeId =
                (Integer) employee.getPropertyValue("EmployeeID");
        stmt.setInt(1, employeeId.intValue());

        ResultSet rs = stmt.executeQuery();

        // For each record in the database create beneficiary entity.
        //
        while (rs.next())
        {
            Entity beneficiary =
                new Entity(employee.getObjectModelId(),
"e:Beneficiary");

            beneficiary.setPropertyValue("BeneficiaryName",
                                         rs.getString("beneficiary"));
            beneficiary.setPropertyValue("RelationshipToSelf",

rs.getString("beneficiary_relationship"));

            beneficiaries.add(beneficiary);
        }

        return beneficiaries;
    }

    /**
     * Retrieves claims from the database.
     *
     * @param employee Employee entity for which to retrieve the claims.
     *
     * @param con Database connection.
     *
     * @return A collection of claims entities for the employee.
     *
     * @exception SQLException If an error occurred while executing SQL
     *            command.
     *
     * @exception ObjectModelException An error occurred while accessing
     *              object model definition.
     *
     * @exception InvalidClassTypeException An error occurred while
accessing
     *                                      object model definition.
     */
    private Collection getClaims(Entity employee, Connection con)
        throws SQLException, ObjectModelException,
InvalidClassTypeException
    {
        Collection claims = new ArrayList();

        // Get the information from the database
```

```
        //
        String sql = "SELECT * FROM HEALTH_CLAIM where EMPLOYEE_ID = ?";
        PreparedStatement stmt = con.prepareStatement(sql);

        Integer employeeId =
                   (Integer) employee.getPropertyValue("EmployeeID");
        stmt.setInt(1, employeeId.intValue());

        ResultSet rs = stmt.executeQuery();

        // For each record in the database create claim entity.
        //
        while (rs.next())
        {
            Entity claim =
                new Entity(employee.getObjectModelId(),
"e:HealthClaim");

            claim.setPropertyValue("PatientName",
rs.getString("Patient"));
            claim.setPropertyValue("ServiceDate",
                                      rs.getDate("Service_Date"));
            claim.setPropertyValue("ClaimStatus",
                                      rs.getString("Claim_Status"));
            claim.setPropertyValue("Charges",
                                   new Integer(rs.getInt("Charges")));
            claim.setPropertyValue("AmountPaid",
                                   new
Integer(rs.getInt("Amount_Paid")));
            claim.setPropertyValue("ClaimNumber",
                                   rs.getString("Claim_Number"));

            claims.add(claim);
        }

        return claims;
    }


    /**
     * Adds a collection of Entities as related entities.
     *
     * @param rootEntity An entity to which add related entities.
     *
     * @param roleName The role name of relation.
     *
     * @param relatedEntities Collection of related entities.
     *
     * @exception ObjectModelException An error occurred while accessing
     *              object model definition.
     *
     */
    private void addRelatedEntities(Entity rootEntity, String roleName,
                                    Collection relatedEntities)
```

```
        throws ObjectModelException
    {
        Iterator it = relatedEntities.iterator();

        while (it.hasNext())
        {
            Entity releatedEntity = (Entity) it.next();
            rootEntity.addRelatedEntity(roleName, releatedEntity);
        }
    }
}
```

Authoring a Calligo Subsection

You need to define variables differently when authoring a Calligo sub-section that is used for a Communicator Component Definition. For each variable, the type `DEFINE @VariableName`, where `VariableName` is a variable name that appears in the `<var-name>` tag in the Data Dictionary Definition.

Note: See the sample below or refer to `DataDict.xml` on the Communicator CD for another sample.

Important: Follow these rules when authoring a sub-section for Communicator:

- Do not omit the `@` character before the variable name.
- Ensure that each variable corresponds to a `<data-element>` in a `<data-package>` in the Data Library Definition.
- Do not define values for variables, as you would for normal Calligo variables. If you define values, Communicator ignores the values and displays a warning.

After defining your variables, you can use the variables as you would in any valid Calligo expression (for example, `TEXT` and `ENDTEXT`).

Example: This is an example of a sub-section for use in Communicator:

```
DEFINE @cstreet
DEFINE @ccity
DEFINE @cstate
DEFINE @cpostalCode
DEFINE @cname
DEFINE @account
TEXT
<cstreet>,
<ccity>,
<cstate>, <cpostalCode>
Re: Account Status
Dear <cname>,
Your account number is <account>.
ENDTEXT
```

Data Dictionary Definition

Ultimately, an external application supplies the data used in letter generation. The Data Dictionary Definition (`DataDict.xml`) simplifies the retrieval of this data. It provides a logical mapping between the variable data elements that are used when generating a letter and the data elements in the external application.

The Data Dictionary Definition places each variable from a CDS file in a `<data-package>`. It also specifies which Object Model Definition and Data Package Custom Handler to use.

Example: This is a sample Data Dictionary Definition:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE data-dictionary SYSTEM "DataDict.dtd">
<data-dictionary>
   <!--
   contains multiple data packages,
   each defines a set of data elements
   -->
   <data-packages>

        <data-package>
         <name>Customer</name>
         <description>This is customer data package </description>
         <custom-handler>
            com.insystems.corr.datadict.CustomHandler.CustomerCH
         </custom-handler>
         <OMDef-URL>
            /com/insystems/corr/datadict/config/OMDefCustomer.xml
         </OMDef-URL>

         <!-- define all of the data elements -->
         <data-elements>
         ...
         </data-elements>
      </data-package>

      <data-package>
         <name>Agent</name>
         <description> This is customer data package </description>
         <custom-handler>
            com.insystems.corr.datadict.CustomHandler.AgentCH
         </custom-handler>
         <OMDef-URL>
            /com/insystems/corr/datadict/config/OMDefAgent.xml
         </OMDef-URL>

         <data-elements>
         ...
         </data-elements>
      </data-package>
            ...
      </data-packages>
```

```
</data-dictionary>
```

Example: This is a complete example of a <data-package>:

```
<data-package>
   <name>Customer</name>
   <description>
      This is customer data package
   </description>
   <custom-handler>
      com.insystems.corr.datadict.CustomHandler.CustomerCH
   </custom-handler>
   <OMDef-URL>
      com/insystems/corr/datadict/config/OMDefCustomer.xml
   </OMDef-URL>

   <data-elements>
      <data-element attribute="isKey">
         <var-name>policyno</var-name>
         <display-name>Customer policy number</display-name>
         <xomp>e:customer/policyno</xomp>
         <note>Please enter policy number</note>
      </data-element>
      <data-element attribute="required">
         <var-name>email</var-name>
         <display-name>Customer email</display-name>
         <xomp>e:customer/email</xomp>
         <note>User must provide an email address</note>
      </data-element>
      <data-element attribute="displayOnly">
         <var-name>cname</var-name>
         <display-name>Customer name</display-name>
         <xomp>e:customer/name</xomp>
      </data-element>
      <data-element attribute="required">
         <var-name>accountNo</var-name>
         <display-name>Customer Account number</display-name>
         <xomp>e:customer/rn:account/AccountNo</xomp>
      </data-element>
      <data-element attribute="optional">
         <var-name>cstreet</var-name>
         <display-name>Customer street</display-name>
         <xomp>e:customer/rn:address/street</xomp>
      </data-element>
      <data-element attribute="required">
         <var-name>ccity</var-name>
         <display-name>Customer City</display-name>
         <xomp>e:customer/rn:address/city</xomp>
      </data-element>
      <data-element attribute="required">
         <var-name>cstate</var-name>
         <display-name>Customer state</display-name>
         <xomp>e:customer/rn:address/state</xomp>
      </data-element>
```

```
      <data-element attribute="required">
         <var-name>cpostalCode</var-name>
         <display-name>Customer Postal Code</display-name>
         <xomp>e:customer/rn:address/postalCode</xomp>
      </data-element>
   </data-elements>
</data-package>
```

Note: Refer to `DataDict.xml` on the Communicator CD for a sample Data Library Definition.

Data Package Tags

<name>

This is a required tag that contains the name of the data package.

<description>

This is a required tag that contains a short description of the data package.

<custom-handler>

This is an optional tag that contains a fully qualified name of the Data Package Custom Handler Java class. If this tag is omitted or has an empty value, all the variables in this Data Package are considered as user-entered values.

<data-elements>

This is a required tag that contains the data elements collection for this data package.

<data-element> attributes and tags

Each `<data-package>` contains `<data-element>` tags that determine how data is retrieved and returned for letter generation.

This section explains the attributes and tags contained in each `<data-element>`.

Example: This is an example of `<data-element>`:

```
<data-elements>
      <data-element attribute="isKey">
         <var-name>policyno</var-name>
         <display-name>Customer policy number</display-name>
         <xomp>e:customer/policyno</xomp>
         <note>Please enter policy number</note>
      </data-element>
      <data-element attribute="required">
         <var-name>email</var-name>
         <display-name>Customer email</display-name>
         <xomp>e:customer/email</xomp>
         <note>User must provide an email address</note>
      </data-element>
```

```
    <data-element attribute="displayOnly">
       <var-name>cname</var-name>
       <display-name>Customer name</display-name>
       <xomp>e:customer/name</xomp>
    </data-element>
    <data-element attribute="required">
       <var-name>accountNo</var-name>
       <display-name>Customer Account number</display-name>
       <xomp>e:customer/rn:account/AccountNo</xomp>
    </data-element>
    <data-element attribute="optional">
       <var-name>cstreet</var-name>
       <display-name>Customer street</display-name>
       <xomp>e:customer/rn:address/street</xomp>
    </data-element>
    <data-element attribute="required">
       <var-name>ccity</var-name>
       <display-name>Customer City</display-name>
       <xomp>e:customer/rn:address/city</xomp>
    </data-element>
    <data-element attribute="required">
       <var-name>cstate</var-name>
       <display-name>Customer state</display-name>
       <xomp>e:customer/rn:address/state</xomp>
    </data-element>
    <data-element attribute="required">
       <var-name>cpostalCode</var-name>
       <display-name>Customer Postal Code</display-name>
       <xomp>e:customer/rn:address/postalCode</xomp>
    </data-element>
</data-elements>
```

<data-element> attributes

Each `<data-element>` has a required `attribute` that determines how a data element appears to a Communicator User when generating a letter, as: *required*, *displayOnly*, *optional*, *hidden*, or *isKey*.

required

This value indicates that letter generation requires a data element value. A Web user will not be able to create letter without specifying the value.

displayOnly

This value indicates that a Web user cannot modify a data element value. The value will be read-only to the user.

optional

This value indicates that a data element value is not required for letter generation. Web users can create letters without specifying the value.

hidden

This value indicates that a data element value is hidden from the user. This is useful for confidential letters.

isKey

This value indicates that a data element is a key element for this package. Therefore, to retrieve data from this package, the user must first specify the key element value.

In this example, `policyno` is the key element value for the Customer package.

<var-name> tag (required)

This tag indicates the name of the variable in the CDS sub-section.

<display-name> tag (required)

The value in this tag appears in the Communicator Web client when a user is creating a definition or generating a letter.

<xomp> tag (required)

This tag indicates where to look in the Object Model Definition for the structure and types of data that the Data Package Custom Handler retrieves from an existing application.

The `<xomp>` value usually consists of an entity and a relative path, separated by a forward slash. Depending on how the Object Model is structured, the relative path may include a role name.

See also *<xomp> and Object Model Definitions* on page 41.

<note> tag (optional)

This tag contains a brief instruction or description of the `<data-element>`. This appears in the Communicator Web client when a user is creating a Definition or generating a letter.

```
<note>Please enter the policy number.</note>
```

Object Model Definitions

The Object Model Definition specifies the structure and type of data in your existing application. It is an XML representation of the data schema in this application.

You need to create one Object Model Definition for each `<data-package>` in the Data Library Definition. Enclose the entire Object Model Definition in an `<om>` tag, with the name of the object in the `id` attribute.

Example: This is an example of an Object Model Definition:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE om SYSTEM "omdef.dtd">
<om id="CustomerModel" schemaVersion="1.1">

   <entity id="e:customer" independent="true">
      <property name="ID" type="java.lang.Integer" />
      <property name="name" type="java.lang.String" />
      <property name="policyno" type="java.lang.Long" />
      <property name="email" type="java.lang.String" />

      <entityRef entityId="e:account"
         roleName="rn:account"
         cardinalityLowerBound="0" cardinalityUpperBound="1"/>
      <entityRef entityId="e:address"
         roleName="rn:address"
         cardinalityLowerBound="0" cardinalityUpperBound="-1"/>
   </entity>

   <entity id="e:account" independent="true">
      <property name="AccountNo" type="java.lang.String"/>
      <property name="AccountHolder" type="java.lang.String" />
      <property name="OpenDate" type="java.util.Date" />
      <property name="AccountType" type="java.lang.String" />
      <property name="Balance" type="java.lang.Double" />
   </entity>

   <entity id="e:address" independent="false">
      <property name="street" type="java.lang.String" />
      <property name="city" type="java.lang.String" />
      <property name="state" type="java.lang.String" />
      <property name="postalCode" type="java.lang.String" />
   </entity>
</om>
```

Note: Refer to `OMDefGroup.xml` on the Communicator CD for a sample Data Dictionary Definition.

<entity> tags

Each Object Model Definition contains entities. Each entity is enclosed in an `<entity>` tag, and has an `id` attribute that specifies an entity in your data schema, for example, an employee or a health claim.

Each `<entity>` tag contains a series of `<property>` tags, and may contain one or more `<entityRef>` tags.

<property> tags

Each entity contains a series of `<property>` tags, each with a `name` and `type` attribute:

- the `name` attribute indicates a field name that the Data Package Custom Handler includes in the data package it returns
- the `type` attribute indicates which Java data type this field should use

<entityRef> tags

This tag indicates a relationship between entities in your application's data schema.

entityID

This attribute indicates the entity

ID of the second entity in the relationship.

The value of `entityID` must always have `e:` as a prefix.

roleName

This attribute indicates the alias to a related entity.

The value of `roleName` must always have *rn:* as a prefix.

cardinalityLowerBound, cardinalityUpperBound

These attributes indicate the number of related entities.

Note: A cardinalityUpperBound attribute with the value of -1 indicates an unlimited number of related entities.

Example: `cardinalityLowerBound=0 cardinalityUpperBound=1`

Indicates that there can be zero or one related entity.

`cardinalityLowerBound=1 cardinalityUpperBound=1`

Indicates that there is exactly one related entity.

`cardinalityLowerBound=1 cardinalityUpperBound=-1`

Indicates that there must be at least one related entity.

Example: In the following `entityRef` tag example, a customer can have at most one account but an unlimited number of addresses:

```
<entity id="e:customer"
   …
   <entityRef entityId="e:account"
      roleName="rn:account"
      cardinalityLowerBound="0"
cardinalityUpperBound="1"/>

   <entityRef entityId="e:address"
      roleName="rn:address"
      cardinalityLowerBound="0"
cardinalityUpperBound="-1"/>

…
</entity>
```

<xomp> and Object Model Definitions

The `<xomp>` value in the Data Dictionary Definition explores the data schema represented in the Object Model Definition. The `<xomp>` value includes

- the `name` attribute of a `<property>` tag inside of an `<entity>`
- the `roleName` and `name` attributes, expressed as a relative path.

Example: An example of a `<xomp>` value is:

```
e:customer/rn:address/street.
```

In this example, `e:customer` is the entity name and `rn:address/street` is the relative path.

Example: Another example of a `<xomp>` value is:

```
e:customer/email
```

In this example, `e:customer` is the entity name and `/email` is the name of the property of the customer entity.

Array Support

This section describes the support of arrays in the Communicator Data Mapper.

If the `<xomp>` tag value points to a property that may have multiple values, Communicator inserts a Calligo array into the resulting generated document. You can therefore use the variables with multiple values in the same way you would use regular Calligo arrays.

Example: This is a partial example of the Customer Data package:

```
<data-element attribute="optional">
   <var-name>cstreet</var-name>
   <display-name>Customer street</display-name>
   <xomp>e:customer/rn:address/street</xomp>
</data-element>

<data-element attribute="required">
   <var-name>ccity</var-name>
   <display-name>Customer City</display-name>
   <xomp>e:customer/rn:address/city</xomp>
</data-element>

<data-element attribute="required">
   <var-name>cstate</var-name>
   <display-name>Customer state</display-name>
   <xomp>e:customer/rn:address/state</xomp>
</data-element>
```

All `xomp` tags point to a property in the address entity which is related to a customer entity with a 0 to -1 cardinality: see *Object Model Definitions* on page 39. A customer may therefore have multiple addresses, and the `cstreet` and `ccity` variables can have multiple values and be inserted into a CDS file in a Calligo array.

Data Package Custom Handler Programmer's Reference

This section is the programmer reference for developing Data Package Custom Handlers for Communicator and contains the following topics:

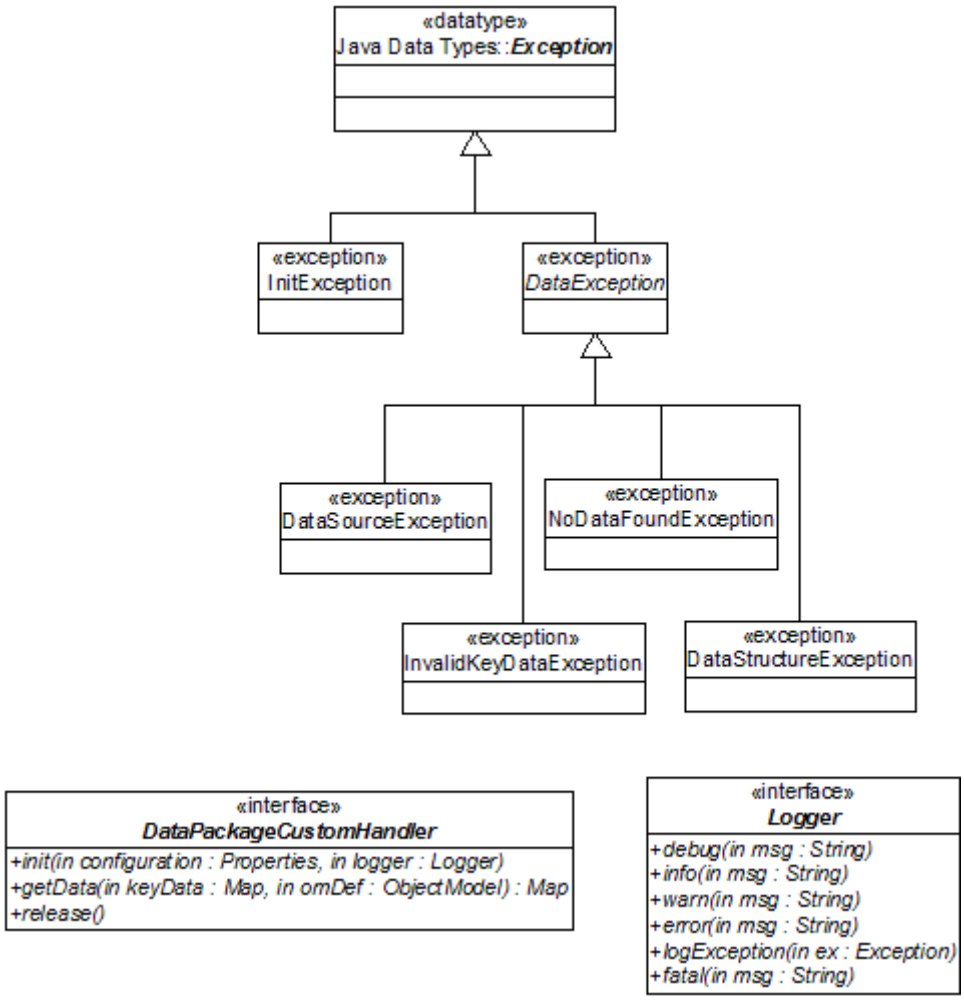
Dependencies

The classes that the Custom Handler Package use depend on the following packages:

Package Name	Location
com.insystems.core.util.converter	Core.jar
com.insystems.objectmodel.eal	ObjectModel.jar
com.insystems.objectmodel.exception	ObjectModel.jar
com.insystems.objectmodel.omdef	ObjectModel.jar
com.insystems.corr.datapackage.customhandler	CorrDPCH.jar

The Package

The `com.insystems.corr.datapackage.customhandler package` contains all the interfaces and supporting classes for developing the Data Package Custom Handler for the Communicator Integrated Data Mapper. (See also *Custom Handler Configuration* on page 46.)



Exceptions for the Custom Handler Package

DataException

Base class for all exceptions thrown by the Data Package Custom Handler, indicating problems with the data.

DataSourceException

An exception indicating a problem with the data source, such as database connectivity or files system problems.

InvalidKeyDataException

An exception indicating a problem with the key data passed to the Custom Handler, such as insufficient data or incorrect data type.

NoDataFoundException

An exception indicating that the custom handler could not find data corresponding to the specified key data values, for example, an incorrect policy number.

DataStructureException

An exception indicating a problem with the data structure, such as `OMDef` and the actual data structure not matching or data manipulation problems.

InitException

An exception indicating a problem with the custom handler initialization, such as being unable to connect to a data source.

DataPackageCustomHandler

The interface between the Communicator Integrated Data Mapper and custom data source. A class specified in the `<custom-handler>` tag in the `DataDict.xml` must implement this interface.

Operations

This interface defines the following operations:

init()

Description

Called by the Communicator Integrated Data Mapper to initialize the Custom Handler.

Code

```
void init(Properties configuration, Logger logger) throws
InitException
```

Parameters

- **configuration** – the configuration parameters: see *Custom Handler Configuration* on page 46 for more details.
- **logger** – an instance of the Logger that Custom Handler must use to log debug and info messages.

getData()

Description

Called by the Communicator Data Library to retrieve the data.

Code

```
Map getData( Map keyValues, ObjectModel omDef) throws
DataException
```

Parameters

- **keyValues** – A map containing key data values. A key to this map is: `String`, which contains the key data name, and `value`, which is the object of the appropriate type holding the key data value.
- **omDef** – Object Model definition.

Returns

A map of entity instances according to the Object Model Definition.

release()

```
void release()
```

Called by the Communicator Data Mapper before the Custom handler is released.

Logger

The interface that provides the logging facility for the Custom Handler. The handler must use this facility to log its messages instead of using:

```
System.out.println()
```

The following operations are defined by this interface:

Operation	Description
void debug(String msg)	Logs a debug level message
void info(String msg)	Logs an information level message
void warn(String msg)	Logs a warning level message
void error(String msg)	Logs an error message
void fatal(String msg)	Logs a fatal level message
long logException(Exception e)	Logs an exception

Custom Handler Configuration

Custom Handler configuration is read from the standard Java properties file located in the same jar as the Custom Handler class. Configuration parameters for the handler may be database name, driver, host name, port, and so on.

For example, if the custom handler class is:

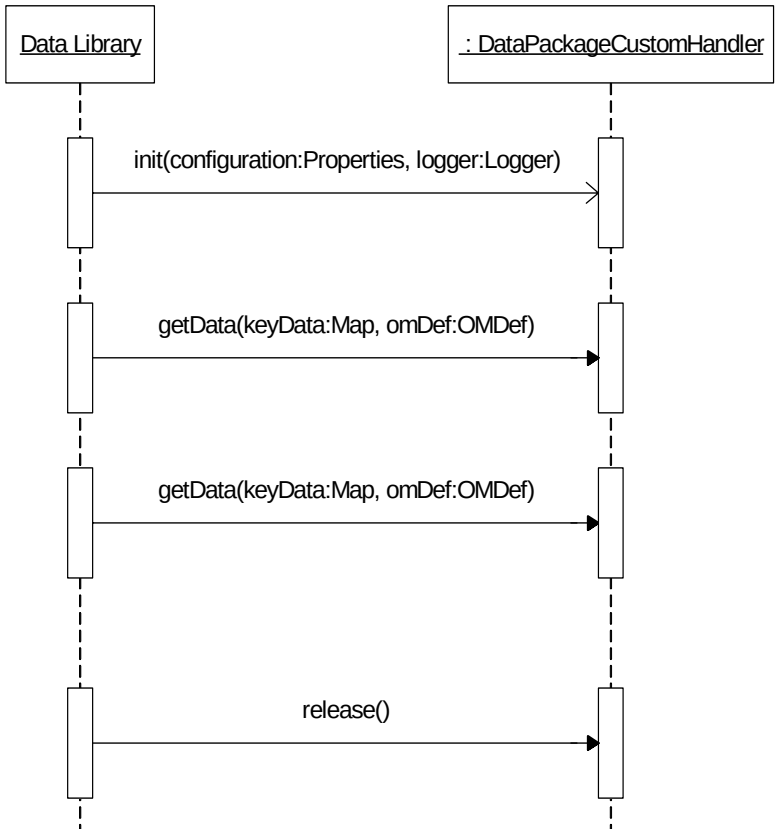
```
com.abccorporation.corr.customhandlers.AccountInfoDPCH
```

and it is in the custom handler is in the `customhandlers.jar`, then the internal structure of the jar file could appear as:

```
...
com/abccorporation/corr/customhandlers/AccountInfoDPCH.class
com/abccorporation/corr/customhandlers/AccountInfoDPCH.properties
...
```

If corresponding properties file is not found, no configuration is read and the Custom Handler receives an empty configuration at initialization.

Sample Sequence Diagram



See also *The Package* on page 43 and *Exceptions for the Custom Handler Package* on page 44.

Chapter 3

Using the SDK

This chapter describes the Software Developers Kit (SDK) and its use. You can use the SDK to extend and customize Communicator's functionality.

This chapter contains information about:

The Communicator SDK

The Communicator SDK is an XML API that enables a third-party software client to generate letters based on existing letter definitions. The SDK provides a loosely coupled enterprise integration of Communicator with third-party custom solutions.

The SDK can be implemented in any language that supports the ORB protocol. The service runs in an EJB container that provides load-balancing and other enterprise features.

XML schema definitions are provided for the Communicator Message and Communicator Message Reply functions. A third-party application can therefore check the validity of the request that it sends and process the response message.

Communicator supports only Letter Generation Requests. A generation request includes all the information needed to generate a letter, such as:

- user credentials
- the path to the Letter Definition
- add-Ins
- key and variable data
- recipient information

Multiple generation requests are allowed in a single XML message. Each XML message should have credentials matching the existing Communicator user.

The Client calls Communicator, passes the XML message, and gets a reply in the form of an XML reply message. The reply message contains the results of all generation requests included in the request message. Each response indicates:

- the outcome of the generation request
- the generated letter identifier if a generation succeeded
- the error code and description if the generation failed

The Structure Tables

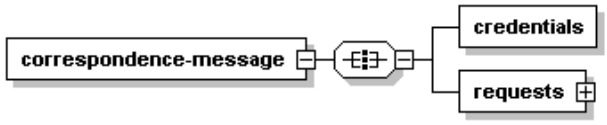
All elements and complex types throughout this guide are described in various tables. Each of these tables contain two or more of the following rows:

- Diagram
- Type
- Facets
- Properties
- Used by element(s)
- Attributes
- Description

Diagram

Every structure table contains a diagram that gives an overview of the structure. Most structures have “child” elements. These will always be to the right of the main element or complex type.

For example, the `correspondence-message` element contains two children: credentials and requests, as the following diagram indicates:



Type

The type of element or complex type.

Facets

The allowed values for the contents of the element or the element’s attribute.

Properties

The properties of the element or complex type.

Used by element(s)

The element(s) that used this element or complex type.

Attributes

The attribute(s) of this element or complex type, including the Name, Type, Use and Description.

Communicator Message Structure

This section describes the Communicator Message structure. Each structure is illustrated in a table containing a structure diagram and other information. For a complete description of these tables, see *The Structure Tables* on page 51.

Note: The namespace for all structures is:

```
http://www.insystems.com/correspondence12
```

The complete source code and schema of Communicator SDK messages is located at:

```
[Install Directory]\Post Installation\SDKExamples
\schema\CorrespondenceMessage.xsd
```

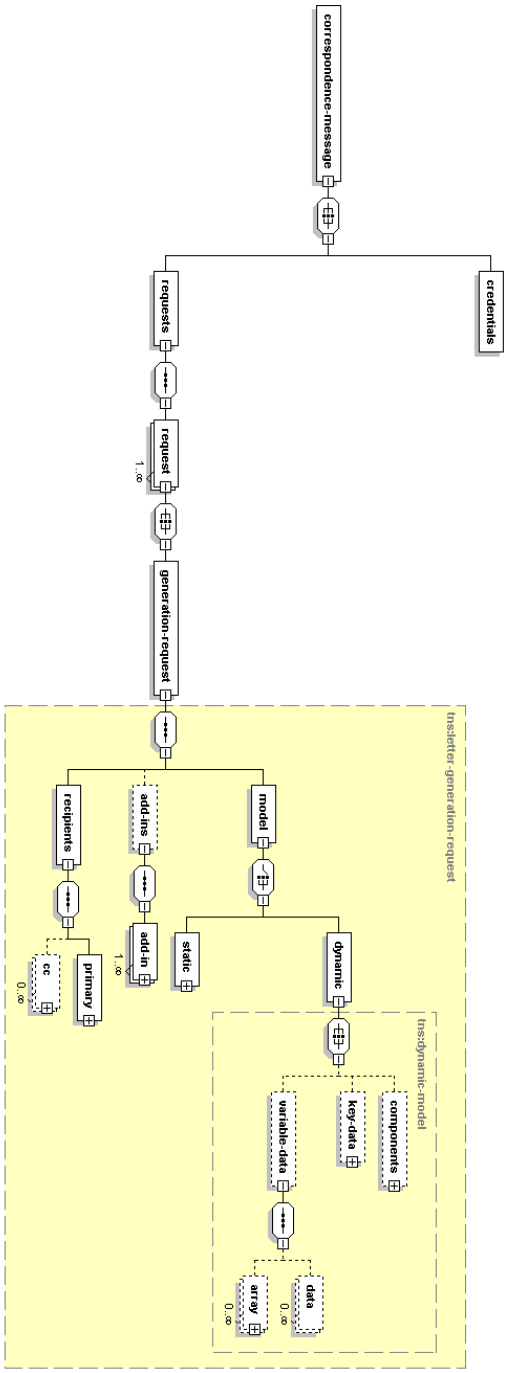
Overview of Message Structure

The Communicator messages consist of multiple requests and one set of credentials with which to log into Communicator.

The basic elements of the message structure are:

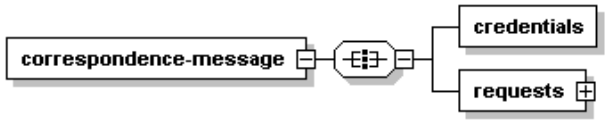
- *correspondence-message element* on page 53: A Communicator API message.
- *correspondence-message/credentials element* on page 54: The credentials of a valid Communicator user.
- *correspondence-message/requests element* on page 54: All requests included in this message.
- *correspondence-message/requests/request element* on page 54: A single request to Communicator.
- *correspondence-message/requests/request/generation-request element* on page 55: The request details. Currently the only supported request type is a generation request.

A diagram showing the overall structure appears on the next page.



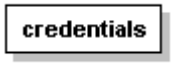
correspondence-message element

Diagram



Description A Communicator API message.

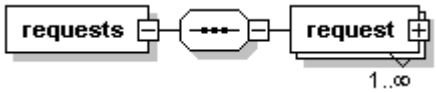
correspondence-message/credentials element

Diagram	

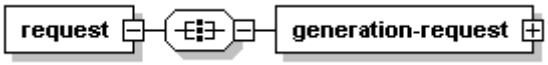
Type	tns:credentials		
Attributes	**name**	**type**	**use**
	username	xs:string	required
	password	xs:string	required
Description	The credentials of a valid Communicator user.		

Important: Multiple requests can be included in a single message. Currently the only supported request type is a generation request. Each request should have an ID.

correspondence-message/requests element

Diagram	
Description	All requests included in this message. Multiple requests can be included in a single message. Each request should have an ID.

correspondence-message/requests/request element

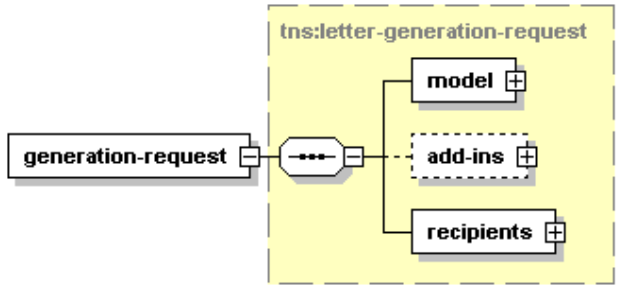
Diagram	

Attributes	**Name**	**Type**	**Use**	**Description**
	id	xs:string	required	Request ID. This ID is used in Communicator response to indicate the request that triggered this response.
Description	A single request to Communicator.			

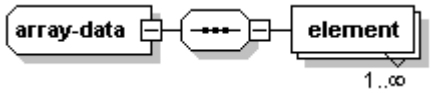
Letter Generation Request

Important: A letter generation request includes a mandatory reference to the model (Letter Definition), recipient information and optional add-ins.

correspondence-message/requests/request/generation-request element

Diagram	
Type	tns:letter-generation-request
Description	The request details. Currently the only supported request type is a generation request.

array-data complexType

Diagram	
Used by	element dynamic-model/variable-data/array

Attributes	Name	Type	Use	Description
	name	xs:string	required	Variable name

array-data/element element

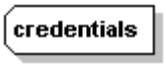
Diagram	

Attributes	Name	Type	Use	Description
	value	xs:string	required	A string representation of the variable value. Communicator will convert this value appropriately.

attribute complexType

Diagram				
Used by elements	recipient/attributes/attribute recipient/distribution-channels/distribution-channel/fields/field			
Attributes	**Name**	**Type**	**Use**	**Description**
	name	xs:string	required	Attribute name
	mapped-value	xs:string	required	String representation of the attribute value. Communicator will convert this value appropriately.
Description	Keeps either mapped value from data dictionary or user-entered value.			

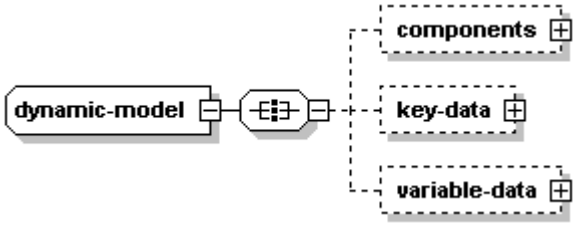
credentials complexType

Diagram			
Used by	element correspondence-message/credentials		
Attributes	**Name**	**Type**	**Use**
	username	xs:string	required
	password	xs:string	required

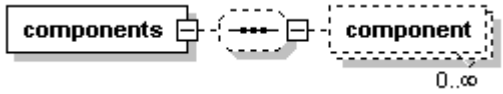
Dynamic Model

A dynamic generation model is used for generating without an existing Calligo Model Section document (CMS). Communicator creates a temporary dynamic CMS.

dynamic-model complexType

Diagram	
Used by	element letter-generation-request/model/dynamic

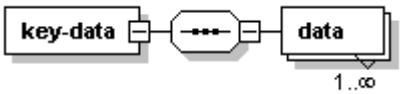
dynamic-model/components element

Diagram				
Attributes	**Name**	**Type**	**Use**	**Description**
	overrideLetterOrder	xs:boolean	optional	Specifies whether the order of components specified in the requests overrides the order of components in the Letter.
	includeOptional	xs:boolean	optional	Specifies whether to include optional Components that are not included explicitly in the request.
Description	Components to include in Letter generation. If “overrideLetterOrder” is set to true, all Components must be specified in the desired order.			

dynamic-model/components/component element

Diagram				
Attributes	**Name**	**Type**	**Use**	**Description**
	name	xs:string	required	Component Definition name
Description	Component to include in generation.			

dynamic-model/key-data element

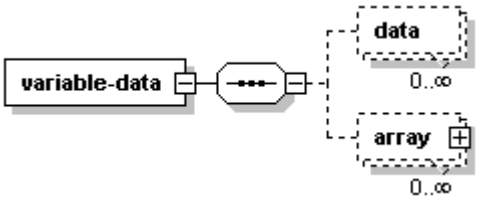
Diagram	
Description	Key Data values required for generation. All key data values must be specified.

Important: Values for all key data specified for a letter definition should be defined data.

dynamic-model/key-data/data element

Diagram				
Type	tns:key-data			
Attributes	**Name**	**Type**	**Use**	**Description**
	name	xs:string	required	Variable name
	value	xs:string	required	String representation of the variable value. Communicator will convert this value appropriately.
Description	Key Data value			

dynamic-model/variable-data element

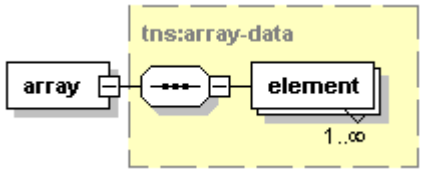
Diagram	
Description	Variable Data values required for generation. Values specified here override values from the Data Dictionary. Values for optional variables may be omitted. Values for required user entered variables must be specified.

Important: Variable data values should be defined for all user-defined data.

dynamic-model/variable-data/data element

Diagram				
Type	tns:variable-data			
Attributes	**Name**	**Type**	**Use**	**Description**
	name	xs:string	required	Variable name
	value	xs:string	required	String representation of the variable value. Communicator will convert this value appropriately.
Description	Variable Data value			

dynamic-model/variable-data/array element

Diagram	

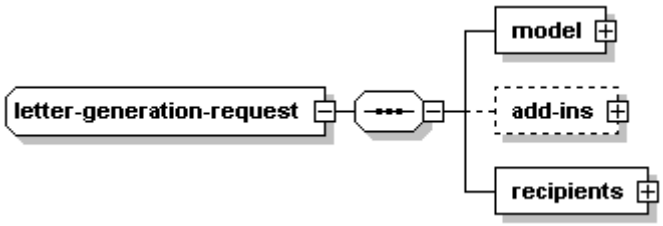
Type	tns:array-data			
Attributes	**Name**	**Type**	**Use**	**Description**
	name	xs:string	required	Variable name
Description	Variable Data array value			

key-data complexType

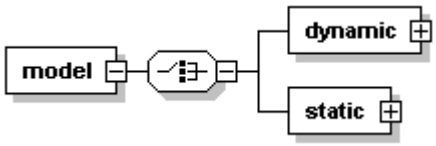
Diagram	
Used by elements	dynamic-model/key-data/data static-model/key-data/data

Attributes	**Name**	**Type**	**Use**	**Description**
	name	xs:string	required	Variable name
	value	xs:string	required	String representation of the variable value. Communicator will convert this value appropriately.

letter-generation-request complexType

Diagram	
Used by	element correspondence-message/requests/request/generation-request
Description	Generation request details

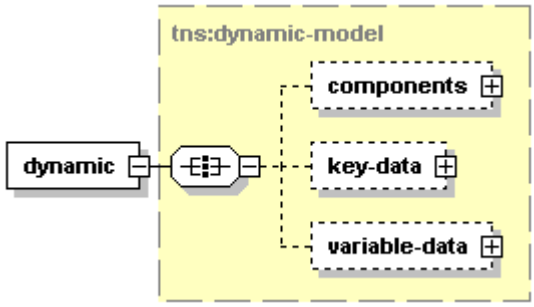
letter-generation-request/model element

Diagram	

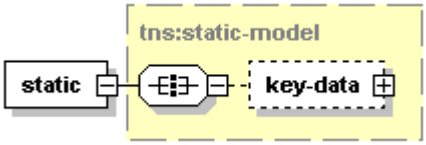
Attributes	Name	Type	Use	Description
	name	xs:string	required	Full path to Letter Definition.

Description	Letter Definition details. Communicator generates a Letter from the definition specified in the “name” attribute.

letter-generation-request/model/dynamic element

Diagram	
Type	tns:dynamic-model
Description	Specifies details of a “dynamic” Letter Definition. Dynamic Letter Definitions are “Fixed” and “Custom” Letter Definitions.

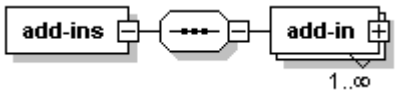
letter-generation-request/model/static element

Diagram	
Type	tns:static-model
Description	Specifies details of a “static” Letter Definition. Static Letter Definitions are based on existing CMS models.

Add-Ins

If an Add-In in the Letter Definition is a Dynamic Add-In, then the location of the document should be defined. For all required Add-Ins, the number of copies should be greater than zero.

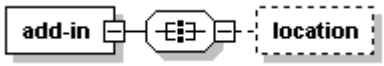
letter-generation-request/add-ins element

Diagram	

Attributes	**Name**	**Type**	**Default**	**Description**
	overrideLetterOrder	xs:boolean	false	Specifies whether the order of add-ins specified in the requests overrides the order of add-ins in Letter Definition.

Description	Add-Ins to include in Letter distribution. If “overrideLetterOrder” attribute is set to true, all Add-Ins must be specified in the desired order.

letter-generation-request/add-ins/add-in element

Diagram	

Attributes	**Name**	**Type**	**Use**	**Description**
	name	xs:string	required	Add-In name
	copies		required	Number of add-in copies to include in distribution.

Description	Add-In to include in distribution.

letter-generation-request/add-ins/add-in/location element

Diagram	

Attributes	**Name**	**Type**	**Use**	**Description**
	repository	xs:string	required	Repository name
	path	xs:string	required	Path to a file within the repository

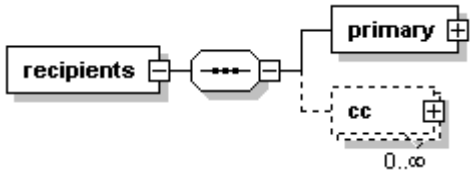
Description	Add-In location for add-ins with no location specified in Attachment.

Recipients

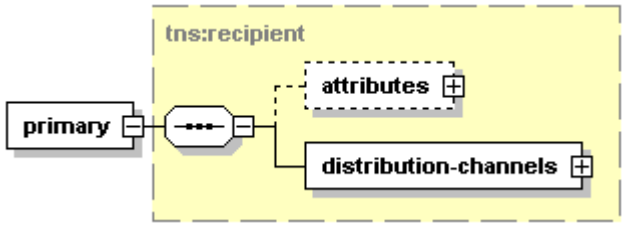
A primary recipient is mandatory. CC recipients are optional and should be selected from the recipients defined in Letter Definition.

Important: Each recipient defines a collection of distribution destinations. Each distribution destination defines a distribution target type (print, email or fax) and collection of parameters.

letter-generation-request/recipients element

Diagram	
Description	Letter recipients

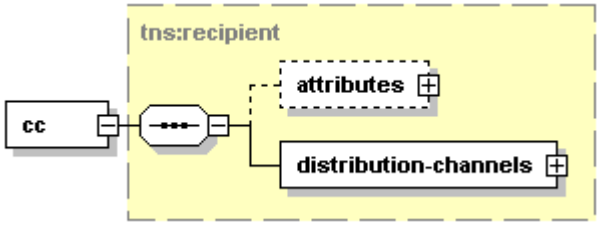
letter-generation-request/recipients/primary element

Diagram	
Type	tns:recipient

Attributes	**Name**	**Type**	**Use**	**Description**
	name	xs:string	required	Recipient “title” as defined in Letter Definition. For primary recipient, this must be “Primary”.

Description	Primary recipient

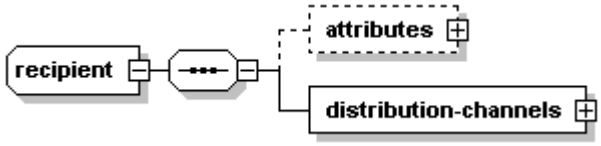
letter-generation-request/recipients/cc element

Diagram	
Type	tns:recipient

Attributes	**Name**	**Type**	**Use**	**Description**
	name	xs:string	required	Recipient “title” as defined in Letter Definition. For primary recipient, this must be “Primary”.

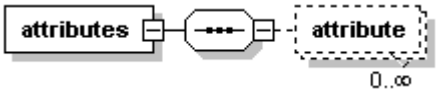
Description	CC Recipient

recipient complexType

Diagram	
Used by elements	letter-generation-request/recipients/cc letter-generation-request/recipients/primary

Attributes	Name	Type	Use	Description
	name	xs:string	required	Recipient “title” as defined in Letter Definition. For primary recipient, this must be “Primary”.

recipient/attributes element

Diagram	
Description	Recipient distribution attributes

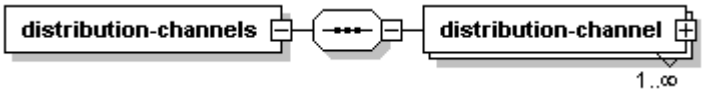
recipient/attributes/attribute element

Diagram	
Type	tns:attribute

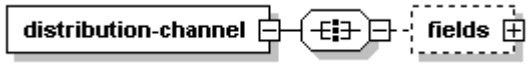
Attributes	Name	Type	Use	Description
	name	xs:string	required	Attribute name.
	mapped-value	xs:string	required	String representation of the attribute value. Communicator will convert this value appropriately.

Description	Distribution attribute value

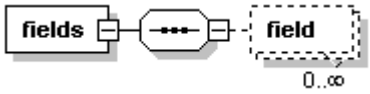
recipient/distribution-channels element

Diagram	
Description	Communicator will distribute the generated letter to all distribution channels specified here.

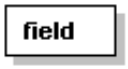
recipient/distribution-channels/distribution-channel element

Diagram				
Attributes	**Name**	**Type**	**Use**	**Description**
	type	xs:string	required	Specifies channel type. Currently supported values are “email”, “fax” and “print”.
Description	Distribution channel			

recipient/distribution-channels/distribution-channel/fields element

Diagram	
Description	Distribution channel attributes

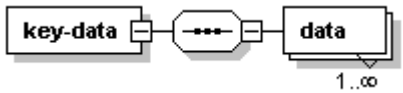
recipient/distribution-channels/distribution-channel/fields/field element

Diagram				
Type	tns:attribute			
Attributes	**Name**	**Type**	**Use**	**Description**
	name	xs:string	required	Attribute name
	mapped-value	xs:string	required	String representation of the attribute value. Communicator will convert this value appropriately.
Description	Distribution attribute value			

static-model complexType

Diagram	
Used by	element letter-generation-request/model/static

static-model/key-data element

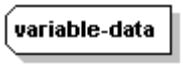
Diagram	
Description	Key Data values required for generation. All key data values must be specified.

Important: Values of all key data used in a Calligo Model Document must be defined.

static-model/key-data/data element

Diagram				
Type	tns:key-data			
Attributes	**Name**	**Type**	**Use**	**Description**
	name	xs:string	required	Variable name
	value	xs:string	required	String representation of the variable value. Communicator will convert this value appropriately.
Description	Key Data value			

complexType variable-data

Diagram				
Used by	element dynamic-model/variable-data/data			
Attributes	**Name**	**Type**	**Use**	**Description**
	name	xs:string	required	Variable name
	value	xs:string	required	String representation of the variable value. Communicator will convert this value appropriately.

Communicator Message Responses

This section explains the Communicator Message response structure. Each section is illustrated in a table containing a structure diagram and other information. For a complete description of these tables, see *The Structure Tables* on page 51.

The response message contains results of all generation requests included in the request message. Each response indicates the outcome of the generation request, the generated letter identifier if a generation succeeded, and the error code and description if the generation failed.

Note: The namespace for all structures is:

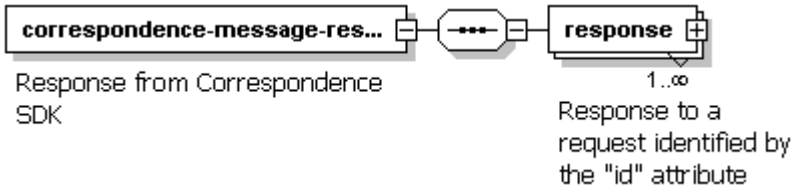
```
http://www.insystems.com/correspondence12
```

The complete source code and schema of Communicator SDK message responses is located at:

[Install Directory]`\Post Installation\SDKExamples\schema\CorrespondenceMessageResponse.xsd`

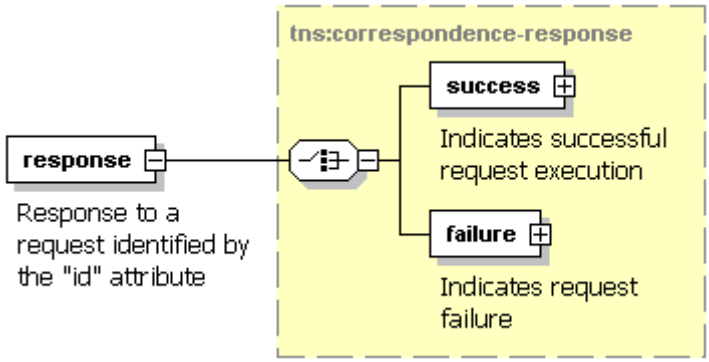
correspondence-message-response element

Diagram



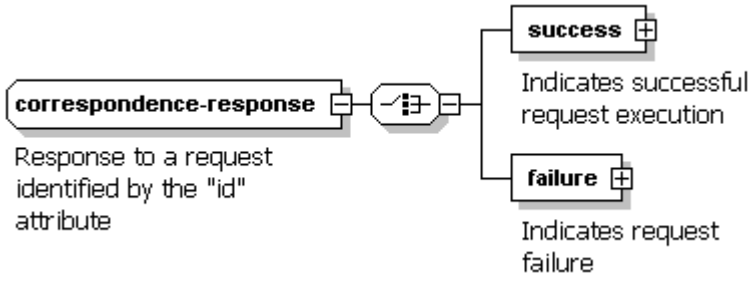
Properties	**isRef**: 0 **content**: complex
Description	Response from Communicator SDK

correspondence-message-response/response element

Diagram	

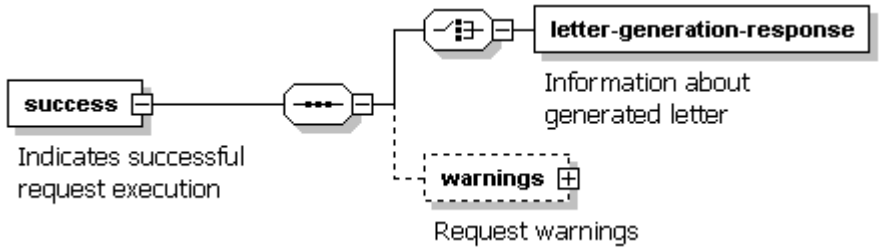
Type	tns:correspondence-response			
Properties	**isRef**: 0 **content**: complex			
Attributes	**Name**	**Type**	**Use**	**Description**
	id	xs:string	required	ID of the request
Description	Response to a request identified by the "id" attribute			

correspondence-response complexType

Diagram	

Used by	element correspondence-message-response/response			
Attributes	**Name**	**Type**	**Use**	**Description**
	id	xs:string	required	ID of the request
Description	Response to a request identified by the "id" attribute			

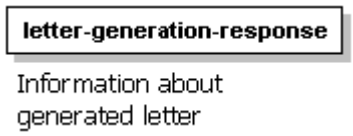
correspondence-response/success element

Diagram	
Properties	**isRef**: 0 **content**: complex
Description	Indicates successful request execution

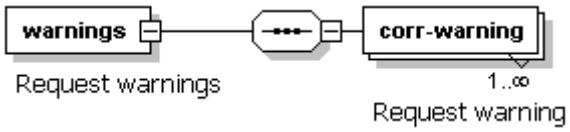
Letter Generation Response

The following elements are included in Letter Generation Response:

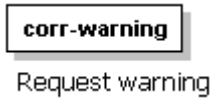
correspondence-response/success/letter-generation-response element

Diagram				
Properties	**isRef**: 0 **content**: complex			
Attributes	**Name**	**Type**	**Use**	**Description**
	letter-id	xs:string	required	Internal Communicator Letter ID
Description	Information about generated letter			

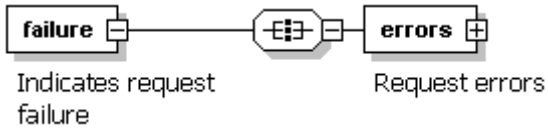
correspondence-response/success/warnings element

Diagram	
Properties	**isRef**: 0 **content**: complex
Description	Request warnings

correspondence-response/success/warnings/corr-warning element

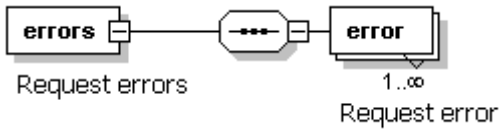
Diagram				
Type	tns:info			
Properties	**isRef**: 0 **content**: complex			
Attributes	**Name**	**Type**	**Use**	**Description**
	code	xs:string	required	Error or Warning code
	description	xs:string	optional	Error or Warning description
Description	Request warning			

correspondence-response/failure element

Diagram	
Properties	**isRef**: 0 **content**: complex
Description	Indicates request failure

Important: One error only is returned if there is a failure in the current implementation.

correspondence-response/failure/errors element

Diagram	
Properties	**isRef**: 0 **content**: complex
Description	Request errors

Important: Info type is used for both errors and warnings.

correspondence-response/failure/errors/error element

Diagram				
Type	tns:info			
Properties	**isRef**: 0 **content**: complex			
Attributes	**Name**	**Type**	**Use**	**Description**
	code	xs:string	required	Error or Warning code
	description	xs:string	optional	Error or Warning description
Description	Request error			

info complexType

Diagram				
Used by elements	correspondence-response/success/warnings/corr-warning correspondence-response/failure/errors/error			
Attributes	**Name**	**Type**	**Use**	**Description**
	code	xs:string	required	Error or Warning code
	description	xs:string	optional	Error or Warning description

Message Example

The following Communicator message example shows how to generate four letters from SDK examples. Each letter is sent to printer (assigned to the SDK user) and to e-mail.

Note: More examples are provided with the sample API client that is part of Communicator.

```
<?xml version="1.0" encoding="UTF-8"?>
<correspondence-message xmlns="http://www.insystems.com/
correspondence12"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.insystems.com/correspondence12
.\CorrespondenceMessage.xsd">
   <credentials username="sdkexample" password="Pass1234"/>
   <requests>

      <request id="GENDYNREQ001">
         <generation-request>
            <model name="/SDK Examples/Dynamic/Simple Fixed">
               <dynamic>
               </dynamic>
            </model>
            <recipients>
               <primary name="Primary">
                  <attributes>
                     <attribute name="PersonName" mapped-
value="Agent_Name"/>
                  </attributes>
                  <distribution-channels>
                     <distribution-channel type="email">
                        <fields>
                           <field name="email" mapped-
value="email@company.com"/>
                        </fields>
                     </distribution-channel>
                  </disitribution-channels>
               </primary>
            </recipients>
         </generation-request>
      </request>

   </requests>
</correspondence-message>
```

Calling the API

The Communicator SDK is exposed as an EJB Stateless Session Bean, which has a single business method called `processXMLMessage()`.

The following sample Java code shows how to create the Session Bean and call the method:

```
import com.insystems.corr.api.ejb.*;
import javax.naming.*;
import javax.rmi.PortableRemoteObject;

String xmlMessage = "put sample xml message here";

Context ctx = new InitialContext();

// look up jndi name
Object ref = ctx.lookup("IStream/Corr/Api/CorrXmlApi");

// cast to Home interface
CorrXmlApiHome  corrXmlApiHome =
(CorrXmlApiHome)PortableRemoteObject.narrow(ref,
CorrXmlApiHome.class);

// create the bean instance
CorrXmlApi  corrXmlApi = corrXmlApiHome.create();

String messageReponse =
corrXmlApi.processXMLMessage(xmlMessage);
```

A sample SDK client is provided with Communicator. The client reads the XML file provided as an argument, sends this XML message to Communicator, and then prints out the reply message.

This sample SDK client with full source code and request examples is located at:

[Install Directory]`\SampleSDKClient`

Submitting Request Samples

Run `SampleSDKClient.bat` to submit request examples using the sample SDK client.

The following command line options are supported:

```
SampleSDKClient.bat [<requestXmlFile>
   [-threads <numberOfThreads> -repeat <repeatRequest>] ]
```

where

- `<requestXmlFile>` is the full path to the XML file containing the Communicator requests. The default value is `\RequestExamples.xml`
- `<numberOfThreads>` is the number of simultaneous threads submitting requests. The default value is 1.

- `<repeatRequest>` is the number of times each thread will submit requests in the `<requestXmlFile>`. The default value is 1.

Examples

- To submit default requests one time:

  ```
  SampleSDKClient.bat
  ```

- To submit default requests ten times:

  ```
  SampleSDKClient.bat .\RequestExamples.xml -threads 1 -
  repeat 10
  ```

- To submit custom requests ten times from ten threads: (The requests in the XML file will be submitted a total of 100 times.)

  ```
  SampleSDKClient.bat MyCustomRequests.xml -threads 10
  -repeat 10
  ```

Passing Key Data Using Cookies

Passing Communicator Key Data values using browser cookies enables a third-party application to populate Key Data values on the Key Data screen during interactive letter creation. The cookie name is not case sensitive and there can be many cookies, one cookie for each variable.

The cookies must be in the following format:

```
Cr_EDV.<Calligo Variable Name>=<value>
```

Example

If a Calligo variable name is `Policy_Number` and the value is `APLA00112344`, then the cookie would be:

```
Cr_EDV.Policy_Number=APLA00112344
```

For an example of a JSP page, locate the `CookieSample/index.jsp` file. This file is located in:

```
<was_home>\AppServer\installedApps\<host>\IStreamCommunicato
r.ear\corr.war\CookieSample/index.jsp
```

You can see a working example of this JPS page by navigating to:

```
http://<Communicator host>/corr/CookieSample
```

The page will work without any changes if the business samples are installed and there is user record with a user name of `user` and a password of `password`.

Integrating with Custom Applications

When a user launches Communicator from a third-party application to create a letter or to continue a **Work In Progress** letter, when the user selects **Release** or **Exit** from these functions, the user will be logged out of Communicator and returned to their custom application. The letter will be released or saved before the window is closed.

To enable this function, include the `leaveOnLetterFinish` parameter (set to `true`) in the URL submitted by the front-end application, for example:

`http://<host>/corr/login/login.do?userName=user&organizationIdString=0&password=p1234&leaveOnLetterFinish=true`

Notes

- the `leaveOnLetterFinish` parameter name is case-sensitive
- the Communicator login username and password are mandatory and should be submitted by the front-end application

Chapter 4

Configuring CorrConfig.xml

This chapter describes how to use and configure `CorrConfig.xml` in order to set up and customize Communicator.

Note: Review the comments within this file for details about each section and how to use them.

IStream Communicator Structure

This section describes the Communicator structure of `CorrConfig.xml`. Each element is illustrated in a table containing a structure diagram and other information. For a complete description of these tables, see *The Structure Tables* on page 51.

Overview of Communicator Structure

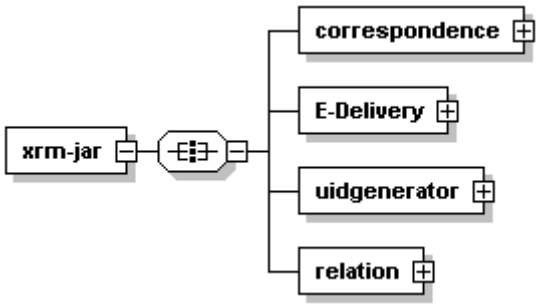
The Communicator element is the highest level, or "root" element of the Communicator structure: for details, see *xrm-jar/correspondence element* on page 79.

This element contains the following secondary elements:

Element and Page Number	Description
xrm-jar/correspondence/calligo-credentials element on page 80	The credentials the Calligo Assembly engine uses to access the Repository and Database InfoSources.
xrm-jar/correspondence/printer-groups element on page 81	The printer groups Communicator uses for printing letters.
xrm-jar/correspondence/options element on page 82	Enables or disables printer and letter disposition override functionality.
xrm-jar/correspondence/max-view-record element on page 82	The number of Letters to show in the Tracking, Work In Progress and Pending Manager's Approval lists.
xrm-jar/correspondence/default-retention-days element on page 83	The number of days that the letter is retained in the temporary retention repository. This allows extra time for the redistribution of letters that were successfully generated and which would otherwise have been deleted.
xrm-jar/correspondence/repositories element on page 83	The repositories Communicator uses.
xrm-jar/correspondence/e-delivery element on page 89	The root element for Communicator settings for Publisher.
xrm-jar/correspondence/distribution element on page 94	The distribution settings.

xrm-jar element

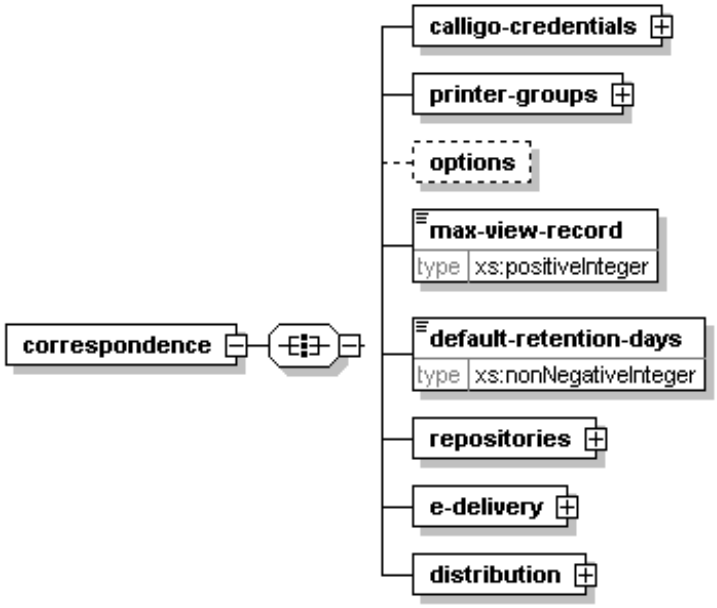
Diagram



Description A root for all configuration.

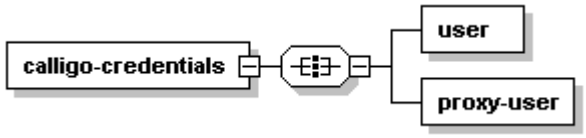
xrm-jar/correspondence element

Diagram

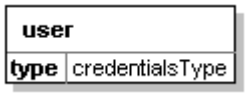


Description The root element for the Communicator-specific configuration.

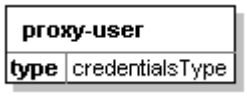
xrm-jar/correspondence/calligo-credentials element

Diagram	
Description	The credentials the Calligo Assembly engine uses to access the Repository and Database InfoSources.

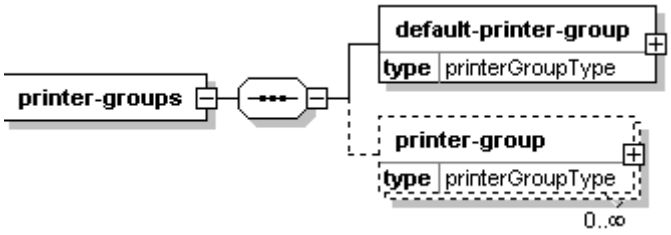
xrm-jar/correspondence/calligo-credentials/user element

Diagram			
Type	credentialsType		
Attributes	**Name**	**Type**	**Use**
	username	xs:string	required
	password	xs:string	required
Description	Calligo User credentials		

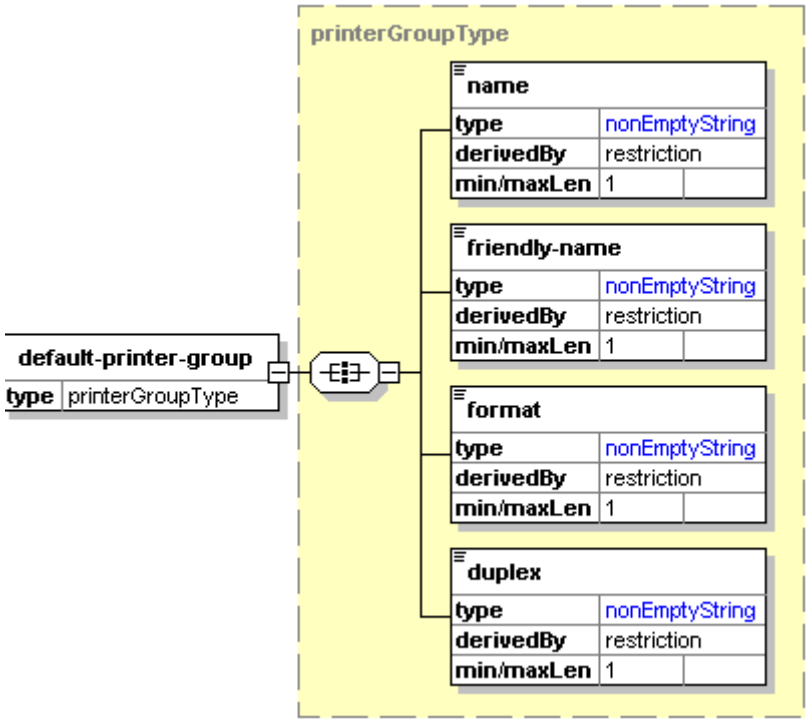
xrm-jar/correspondence/calligo-credentials/proxy-user element

Diagram			
Type	credentialsType		
Attributes	**Name**	**Type**	**Use**
	username	xs:string	required
	password	xs:string	required
Description	Calligo Proxy-User credentials		

xrm-jar/correspondence/printer-groups element

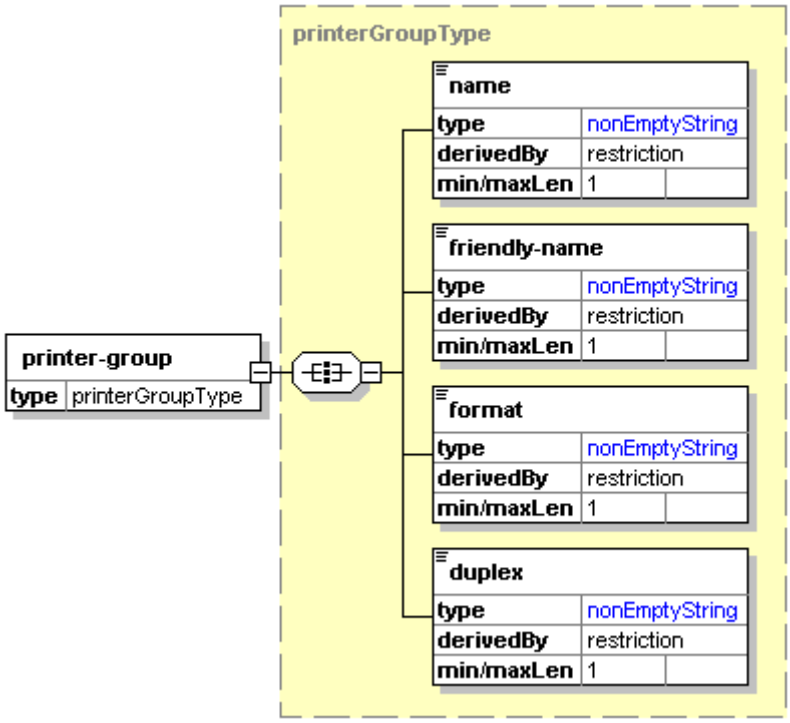
Diagram	
Description	The printer groups Communicator uses for printing letters.

xrm-jar/correspondence/printer-groups/default-printer-group element

Diagram	
Type	printerGroupType
Description	The default printer group. If there is no printer group selected for a Letter Definition or User, a letter will be printed on this printer.

Important: Ensure that the printer is correctly specified by the `name`. If you delete or rename a printer in a printer group, the default printer will be used.

xrm-jar/correspondence/printer-groups/printer-group element

Diagram	
Type	printerGroupType
Description	The printer group available for selection by the Contributor.

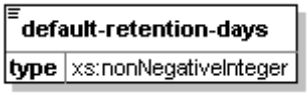
xrm-jar/correspondence/options element

Diagram			
Attributes	**Name**	**Type**	**Use**
	showOverride	xs:boolean	optional
Description	Enables or disables printer and letter disposition override functionality.		

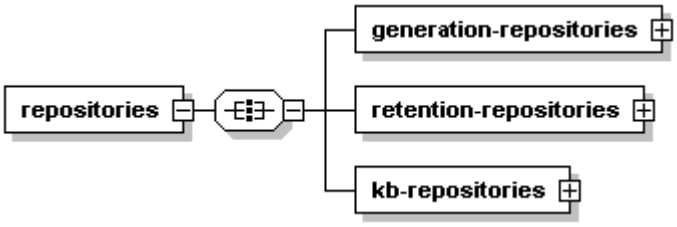
xrm-jar/correspondence/max-view-record element

Diagram	
Type	xs:positiveInteger
Description	The number of Letters to show in the Tracking, Work In Progress and Pending Manager's Approval lists.

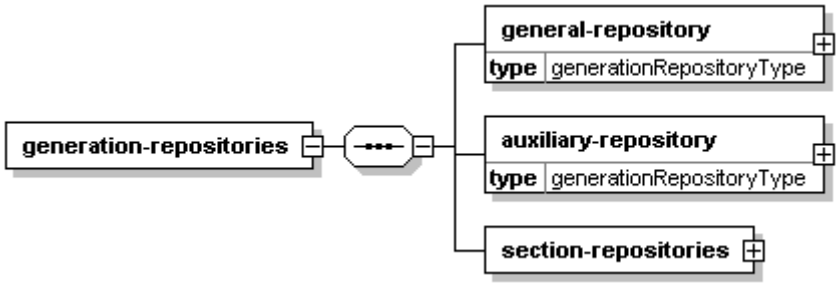
xrm-jar/correspondence/default-retention-days element

Diagram	
Type	xs:nonNegativeInteger
Description	The number of days that the letter is retained in the temporary retention repository. This allows extra time for the redistribution of letters that were successfully generated and which would otherwise have been deleted. **Note:** Redistribution only occurs from this temporary repository. Once a letter is removed, it cannot be redistributed through Communicator. If the letter is in Work in Progress or was not distributed successfully, it will not be deleted from the temporary repository until there is a successful distribution.

xrm-jar/correspondence/repositories element

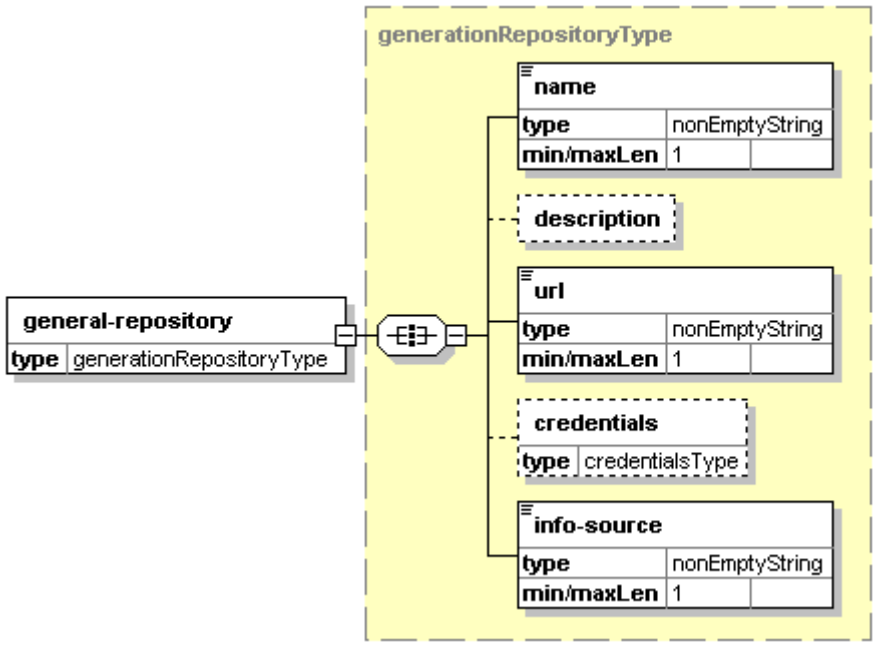
Diagram	
Description	The repositories Communicator uses.

xrm-jar/correspondence/repositories/generation-repositories element

Diagram	
Description	Generation repositories are used for letter generation. These repositories are accessed by Communicator and Calligo Assembly.

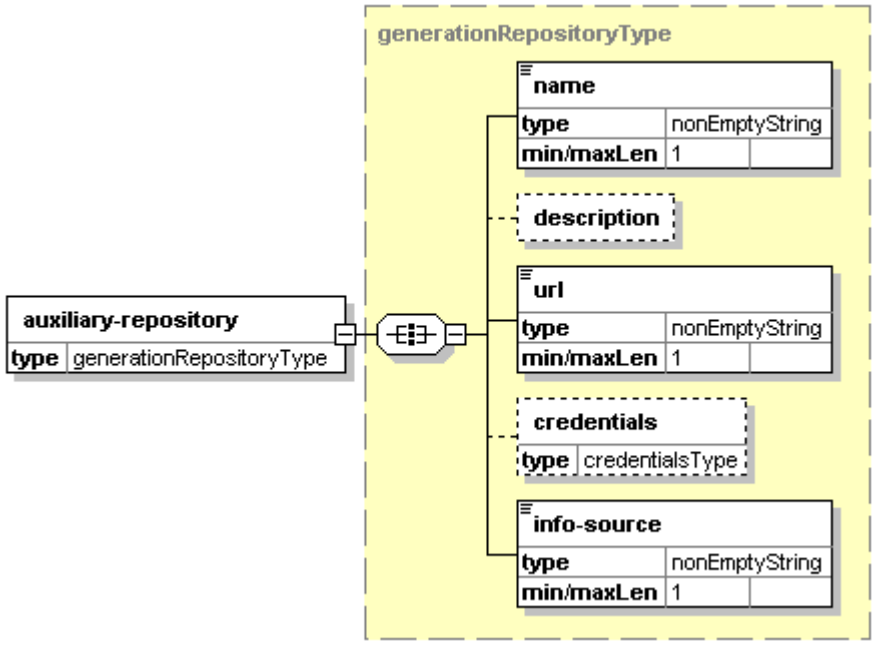
xrm-jar/correspondence/repositories/generation-repositories/general-repository element

Diagram

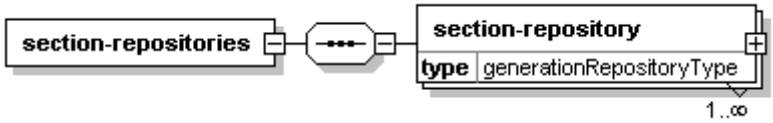


Type	generationRepositoryType
Description	The general repository is used for exchanging generated documents between Communicator, Publisher and Calligo Assembly.

xrm-jar/correspondence/repositories/generation-repositories/auxiliary-repository element

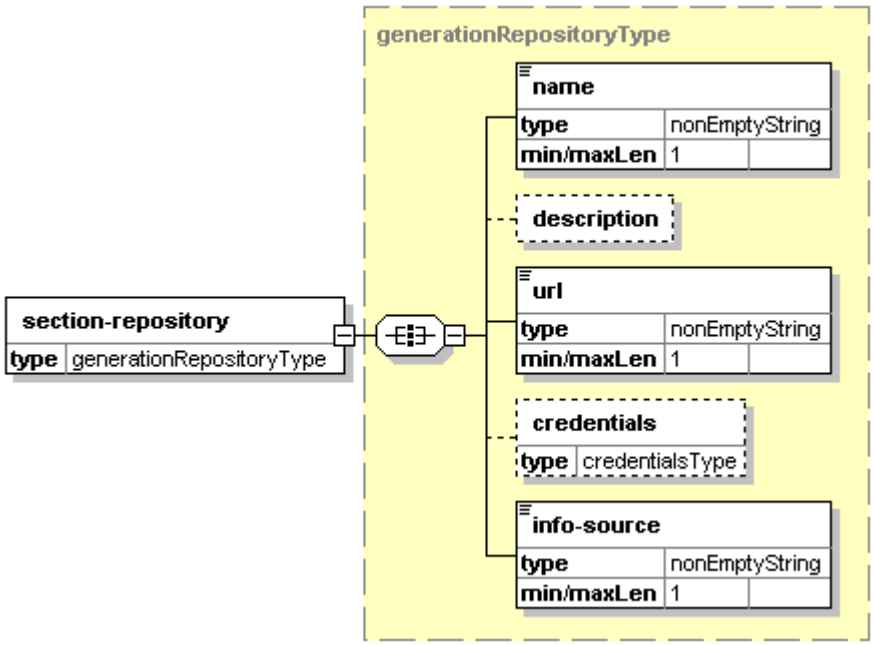
Diagram	
Type	generationRepositoryType
Description	Auxiliary repository stores Calligo Sections used internally by Communicator.

xrm-jar/correspondence/repositories/generation-repositories/section-repositories element

Diagram	
Description	Section repositories are used to store and access Calligo Section that are included in the letter generation. Component definition versions will reference Calligo model document sections (CDS files) stored in these repositories.

xrm-jar/correspondence/repositories/generation-repositories/section-repositories/section-repository element

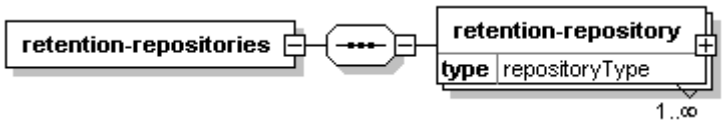
Diagram



Type	generationRepositoryType

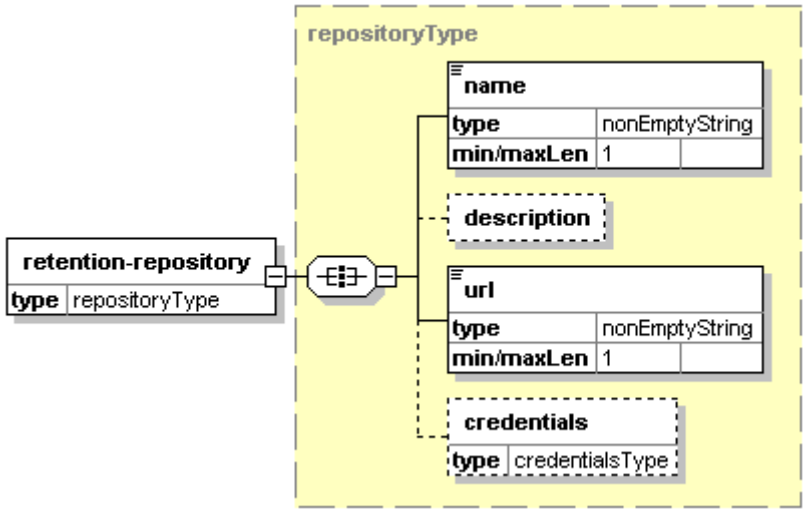
xrm-jar/correspon/repositories/retention-repositories element

Diagram

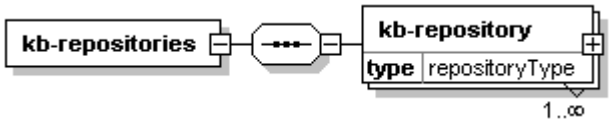


Description	These repositories are used for retaining letters generated by Communicator.

xrm-jar/correspondence/repositories/retention-repositories/retention-repository element

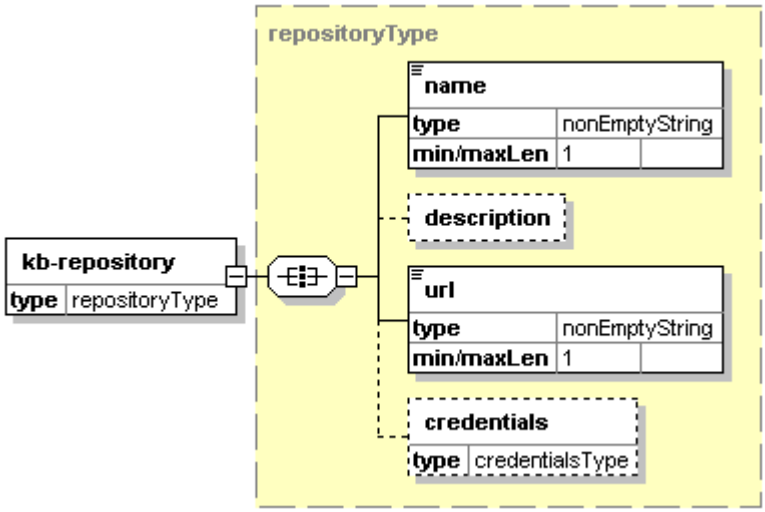
Diagram	
Type	repositoryType

xrm-jar/correspondence/repositories/kb-repositories element

Diagram	
Description	Knowledge Base repositories store Knowledge Base articles associated with Letter Definitions.

xrm-jar/correspondence/repositories/kb-repositories/kb-repository element

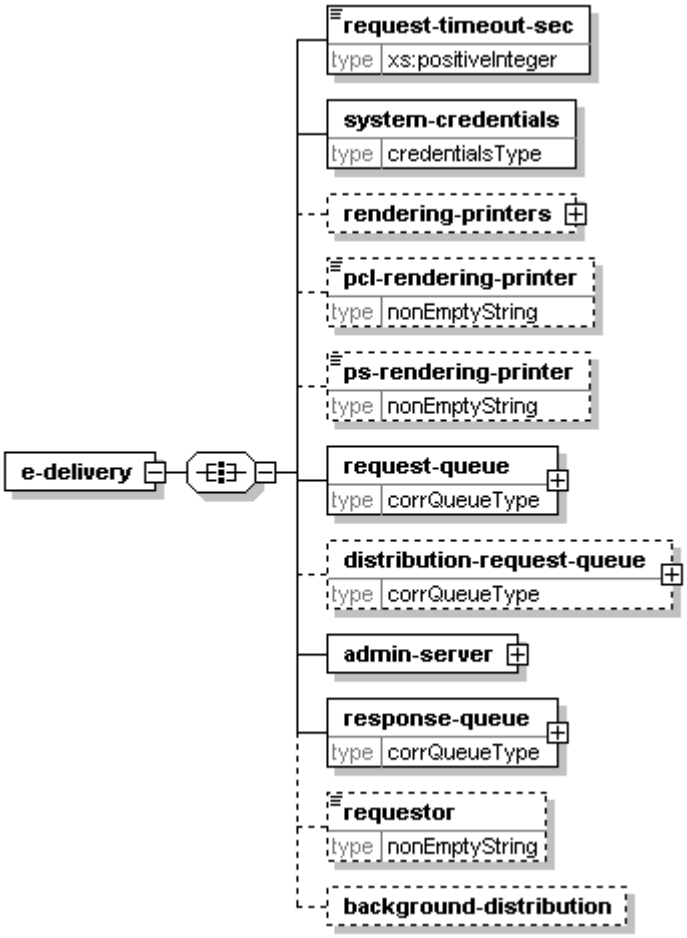
Diagram



Type	repositoryType

xrm-jar/correspondence/e-delivery element

Diagram



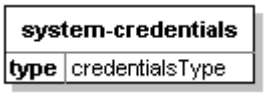
Description	The root element for Communicator settings for Publisher.

xrm-jar/correspondence/e-delivery/request-timeout-sec

Diagram

Type	xs:positiveInteger
Description	The Publisher request timeout in seconds. This parameter defines the maximum time required for the generation of one letter.

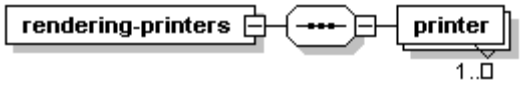
xrm-jar/correspondence/e-delivery/system-credentials element

Diagram	
Type	credentialsType

Attributes	**Name**	**Type**	**Use**
	username	xs:string	required
	password	xs:string	required

Description	The user name and password of the Communicator user who updates the status of the distributed letters. This can be any valid Communicator user.

xrm-jar/correspondence/e-delivery/rendering-printers element

Diagram	
Description	Defines printers for rendering to various printer stream formats, for example, Postscript or PCL. This tag essentially replaces the <pcl-rendering-printer> and <ps-rendering-printer> tags. These tags are still supported for backward compatibility, but we recommend using the new tag. If both the old and new tags are present, the new tags take precedence.

xrm-jar/correspondence/e-delivery/rendering-printers/printer element

Diagram	

Attributes	**Name**	**Type**	**Use**	**Description**
	format	xs:string	required	Mime type, or PCL or Postscript.
	name	xs:string	required	Local printer name

Description	The name of the local printer installed on the Publisher Worker. This printer is used for rendering documents to the required format.

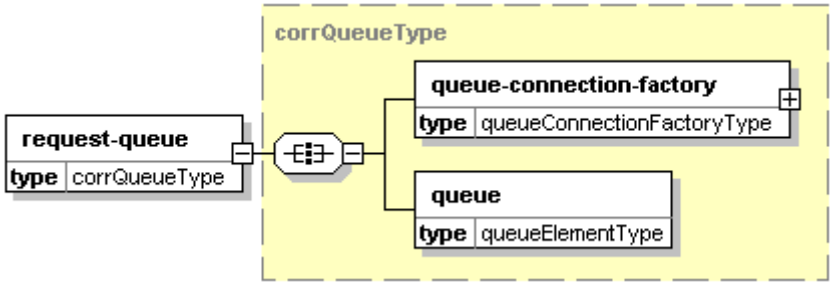
xrm-jar/correspondence/e-delivery/pcl-rendering-printer element

Diagram	
Description	Name of the local printer installed on the Publisher Worker. This printer is used for rendering documents to PCL format.

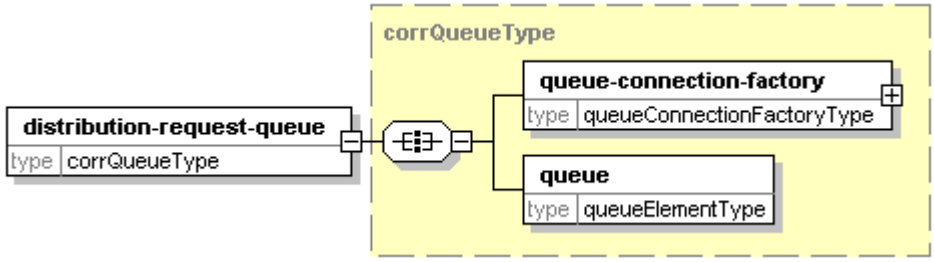
xrm-jar/correspondence/e-delivery/ps-rendering-printer element

Diagram	
Description	Name of the local printer installed on the Publisher Worker. This printer is used for rendering documents to PostScript format.

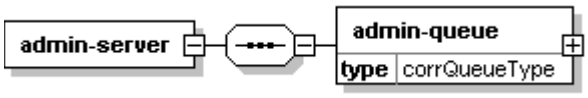
xrm-jar/correspondence/e-delivery/request-queue element

Diagram	
Type	corrQueueType
Description	The Publisher submission queue Communicator uses to submit generation and distributions requests.

xrm-jar/correspondence/e-delivery/distribution-request-queue element

Diagram	
Type	corrQueueType
Description	The Publisher submission queue used by Communicator to submit distributions requests. If omitted, Communicator uses <request-queue> for both generation and distribution requests.

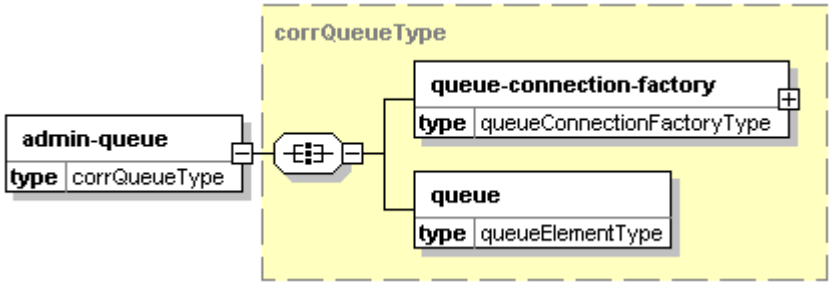
xrm-jar/correspondence/e-delivery/admin-server element

Diagram	

Attributes	**Name**	**Type**	**Use**
	class	xs:string	required
	asynchronous	xs:boolean	required
	timeout	xs:int	required
	mainMBean	xs:string	required

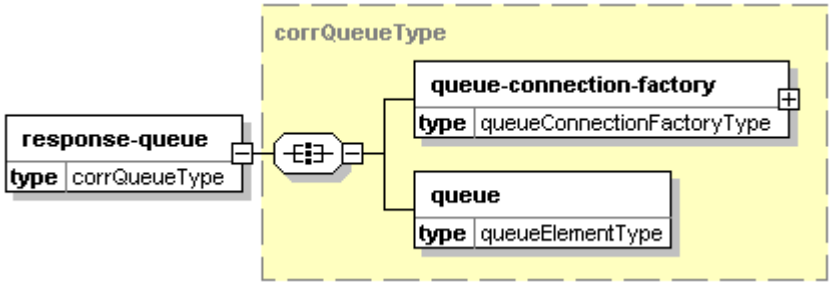
Description	This internal configuration is updated by the configuration wizard. Do not manually edit it.

xrm-jar/correspondence/e-delivery/admin-server/admin-queue element

Diagram	

Type	corrQueueType
Description	The Publisher external administration queue Communicator uses to submit Letter recall requests

xrm-jar/correspondence/e-delivery/response-queue element

Diagram	

Type	corrQueueType
Description	The Publisher response queue Communicator uses to receive service responses.

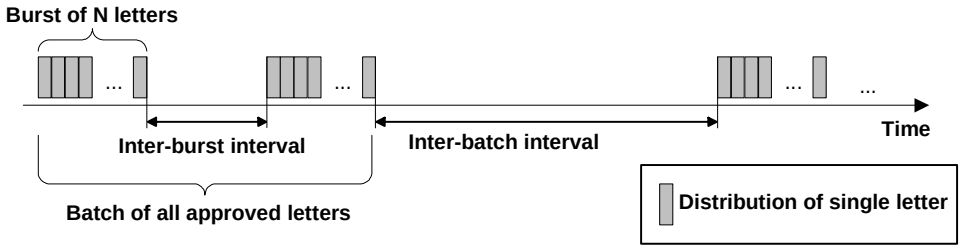
xrm-jar/correspondence/e-delivery/requestor

Diagram	
Description	Communicator populates the `requestor` field of all Publisher requests with the string specified in the body of this element.

xrm-jar/correspondence/e-delivery/background-distribution

This tag controls the parameters for the background distribution of mass approved letters.

The following diagram illustrates how letters are distributed in the background:

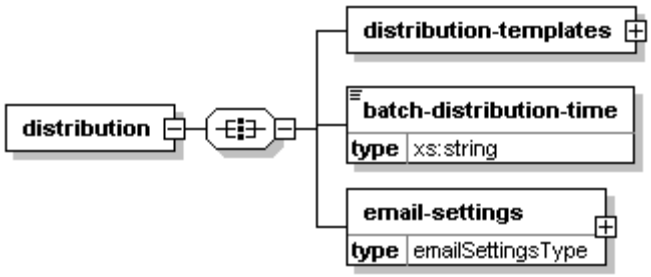


Diagram

Attributes	Name	Type	Use	Default	Description
	inter-burst-interval	xs:decimal	optional	20	Delay between bursts in seconds
	burst-size	xs:decimal	optional	50	Number of letters in one burst (sent out without delays between letters)
	inter-batch-interval	xs:decimal	optional	50	Delay between DB lookups for marked letters in seconds

xrm-jar/correspondence/distribution element

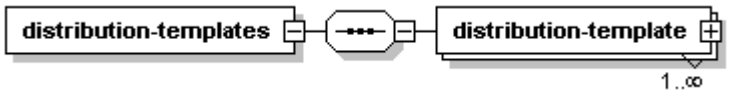
Diagram



Description The distribution settings.

xrm-jar/correspondence/distribution/distribution-templates element

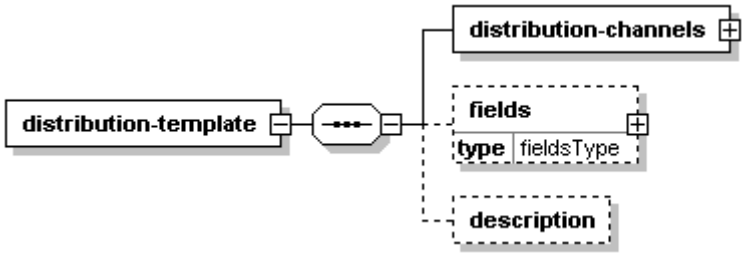
Diagram



Description Distribution templates

xrm-jar/correspondence/distribution/distribution-templates/ distribution-template element

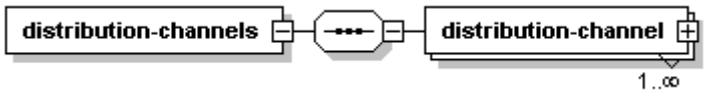
Diagram



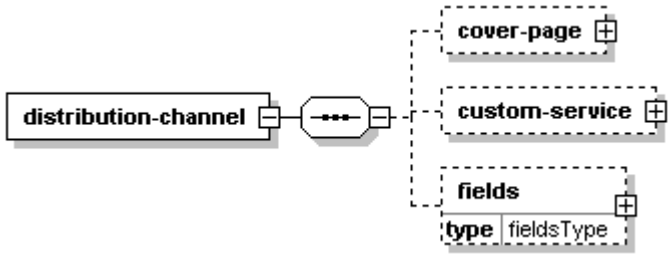
Attributes	Name	Type	Use	Description
	default	xs:boolean	required	Specifies whether this is a default distribution template.
	name	xs:string	required	Distribution template name

Description The distribution template available for selection by Contributor. Each template defines which distribution channels are available and what information is required for distribution.

xrm-jar/correspondence/distribution/distribution-templates/distribution-template/distribution-channels element

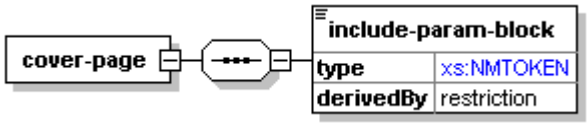
Diagram	
Description	The distribution channels available in the template.

xrm-jar/correspondence/distribution/distribution-templates/distribution-template/distribution-channels/distribution-channel element

Diagram	

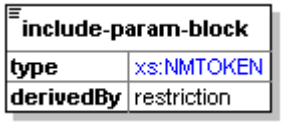
Attributes	**Name**	**Type**	**Use**	**Description**
	type		required	The distribution channel type supported by Publisher.
	secure	xs:boolean	required	Specifies whether the channel can be used for delivering confidential information. If this is false, this channel will not be available for confidential letters.
Description	Individual distribution channel			

xrm-jar/correspondence/distribution/distribution-templates/distribution-template/distribution-channels/distribution-channel/cover-page element

Diagram	

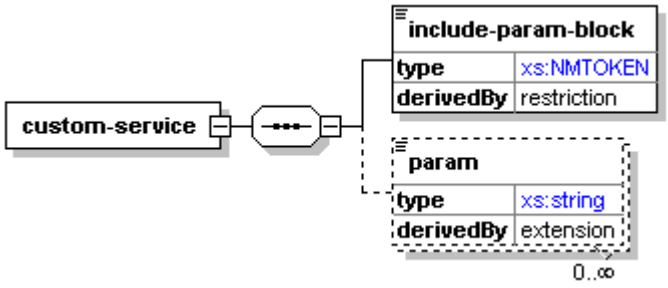
Attributes	**Name**	**Type**	**Use**	**Description**
	uisr	xs:string	required	Calligo UISR of the Cover Page model.
	username	xs:string	required	Calligo username.
	password	xs:string	required	Calligo password.
Description	Cover page information. The cover page will be generated before distributing the letter.			

xrm-jar/correspondence/distribution/distribution-templates/distribution-template/distribution-channels/distribution-channel/cover-page/include-param-block element

Diagram	
Type	restriction of xs:NMTOKEN
Values	**all** – include all data available in Letter as parameters to the Cover Page **none** – the Cover Page does not require any custom data
Keywords	This field can have a combination of one or more keywords, separated by a comma: **ChannelAttributes** – include Distribution Channel attributes **RecipientAttributes** – include Recipient attributes **BusinessData** – include variable data definitions such as type and default values **VariableData** – include variable data values
Facets	enumeration all enumeration none
Description	Specifies what information to pass to Cover Page generation.

xrm-jar/correspondence/distribution/distribution-templates/ distribution-template/distribution-channels/distribution-channel/ custom-service element

Diagram

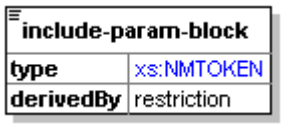


Attributes	Name	Type	Use	Description
	name	xs:string	required	The Publisher Custom Service Name to invoke before distribution.
	class	xs:string	required	

Description The Publisher Custom Service information. If specified, the Publisher Custom Service will be invoked before distribution.

xrm-jar/correspondence/distribution/distribution-templates/ distribution-template/distribution-channels/distribution-channel/ custom-service/include-param-block element

Diagram

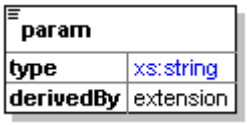


Type restriction of xs:NMTOKEN

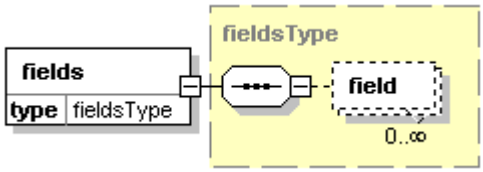
Facets enumeration all
enumeration none

Description Specifies what information to pass to Custom Service invocation.

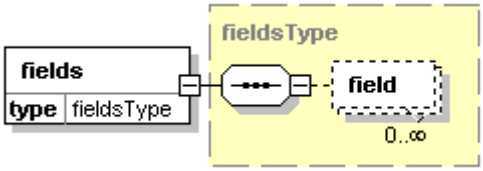
xrm-jar/correspondence/distribution/distribution-templates/distribution-template/distribution-channels/distribution-channel/custom-service/param element

Diagram	
Type	extension of xs:string
Attributes	**Name** / **description** name / Parameter name
Description	A custom service parameter value specified as tag content.

xrm-jar/correspondence/distribution/distribution-templates/distribution-template/distribution-channels/distribution-channel/fields element

Diagram	
Type	fieldsType
Description	Distribution channel fields. The values for these fields are required for the associated distribution channel.

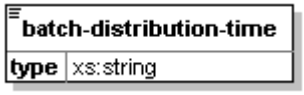
xrm-jar/correspondence/distribution/distribution-templates/distribution-template/fields element

Diagram	
Type	fieldsType
Description	Distribution template fields. The values for these fields are required for all distribution channels.

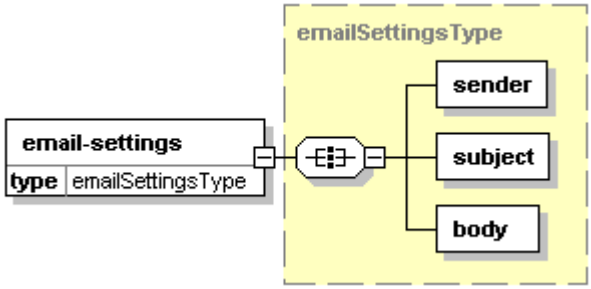
xrm-jar/correspondence/distribution/distribution-templates/ distribution-template/description element

Diagram	
Description	Distribution template description.

xrm-jar/correspondence/distribution/batch-distribution-time element

Diagram	
Type	xs:string
Description	Batch distribution time in the HH:MM:SSAM format.

xrm-jar/correspondence/distribution/email-settings element

Diagram	
Type	emailSettingsType
Description	System wide Email distribution settings

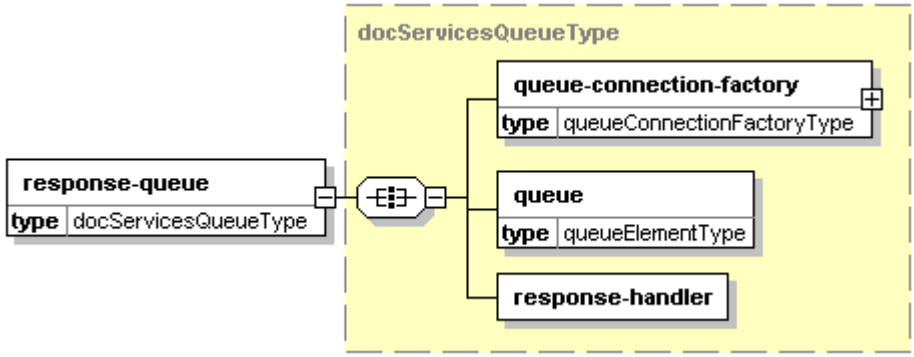
xrm-jar/E-Delivery element

Diagram	
Description	Root element for the Publisher client configuration

xrm-jar/E-Delivery/client element

Diagram	
Description	The Publisher Client configuration.

xrm-jar/E-Delivery/client/response-queue element

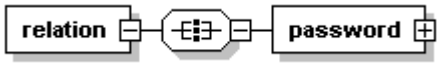
Diagram	

Type	docServicesQueueType		
Attributes	**Name**	**Type**	**Use**
	numberOfListeners	xs:positiveInteger	required
Description	The Publisher response queue Communicator uses to receive distribution responses. The name of this queue must be the same as the one specified in xrm-jar/correspondence/e-delivery/response-queue element.		

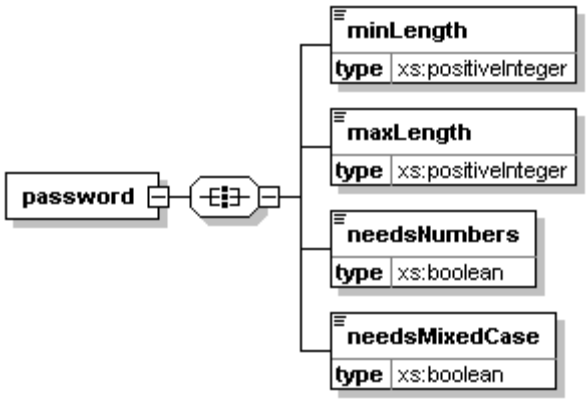
xrm-jar/uidgenerator element

Diagram	
Description	This element contains internal application configuration and is updated by the Communicator Configuration Wizard.

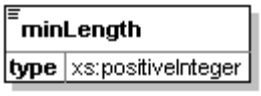
xrm-jar/relation element

Diagram	
Description	Additional Communicator settings

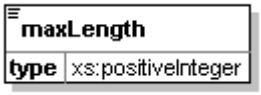
xrm-jar/relation/password element

Diagram	
Description	Password policy configuration

xrm-jar/relation/password/minLength element

Diagram	
Type	xs:positiveInteger
Description	Minimum password length

xrm-jar/relation/password/maxLength element

Diagram	
Type	xs:positiveInteger
Description	Maximum password length

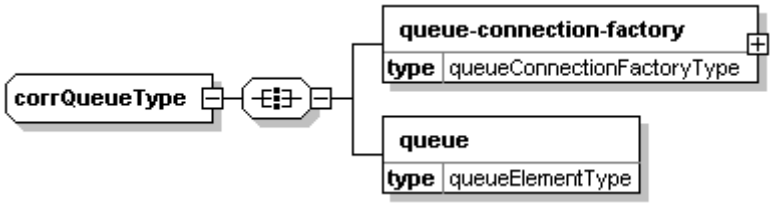
xrm-jar/relation/password/needsNumbers element

Diagram	
Type	xs:boolean
Description	Specifies whether the password must contain numbers.

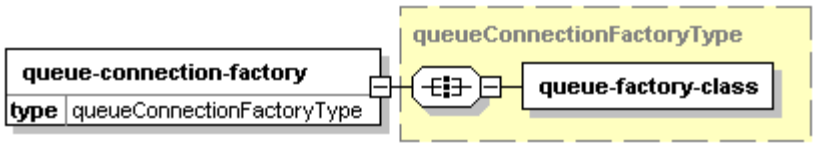
xrm-jar/relation/password/needsMixedCase element

Diagram	
Type	xs:boolean
Description	Specifies whether the password must be in mixed case.

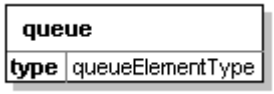
corrQueueType complexType

Diagram	
Used by elements	xrm-jar/correspondence/e-delivery/admin-server/admin-queue xrm-jar/correspondence/e-delivery/request-queue

corrQueueType/queue-connection-factory element

Diagram	
Type	queueConnectionFactoryType
Description	JMS Server Connection parameters

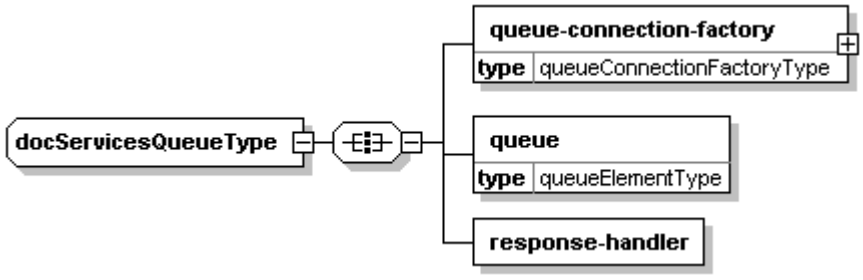
corrQueueType/queue element

Diagram				
Type	queueElementType			
Attributes	**Name**	**Type**	**Use**	**Description**
	type	xs:string	required	Specifies whether the “name” attribute specifies the JMS Server Queue Name or the JNDI context name.
	name	xs:string	required	JMS Server Queue Name or JNDI context name. See “type” attribute.
	user	xs:string	optional	The user name for connection to the JMS Server Queue.
	password	xs:string	optional	The password for connection to the JMS Server Queue.
Description	JMS Server Queue parameters			

credentialsType complexType

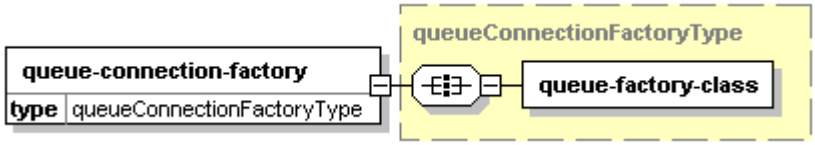
Diagram			
Used by elements	generationRepositoryType/credentials repositoryType/credentials xrm-jar/correspondence/calligo-credentials/proxy-user		
Attributes	**Name**	**Type**	**Use**
	username	xs:string	required
	password	xs:string	optional

docServicesQueueType complexType

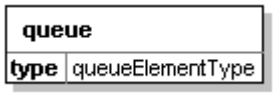
Diagram	

Used by	element xrm-jar/E-Delivery/client/response-queue		
Attributes	**Name**	**Type**	**Use**
	numberOfListeners	xs:positiveInteger	required

docServicesQueueType/queue-connection-factory element

Diagram	
Type	queueConnectionFactoryType
Description	JMS Server Connection parameters

docServicesQueueType/queue element

Diagram	
Type	queueElementType

Attributes	**Name**	**Type**	**Use**	**Description**
	type	xs:string	required	Specifies whether the “name” attribute specifies the JMS Server Queue Name or the JNDI context name.
	name	xs:string	required	JMS Server Queue Name or JNDI context name. See “type” attribute.
	user	xs:string	optional	Username for connection to the JMS Server Queue.
	password	xs:string	optional	Password for connection to the JMS Server Queue.

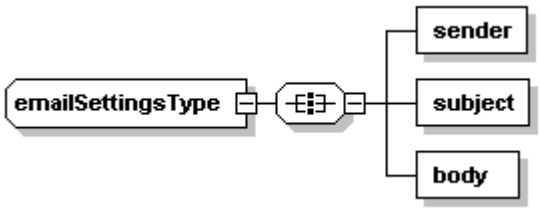
Description	JMS Server Queue parameters

docServicesQueueType/response-handler element

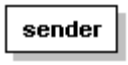
Diagram	

Attributes	**Name**	**Type**	**Use**
	class	xs:string	required
Description	This internal configuration is updated by the configuration wizard. Do not edit manually.		

emailSettingsType complexType

Diagram	
Used by	element xrm-jar/correspondence/distribution/email-settings

emailSettingsType/sender element

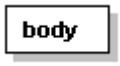
Diagram	

Attributes	**Name**	**Type**	**Use**	**Description**
	email	nonEmptyString	required	Sender email
	name	xs:string	optional	Sender name
Description	The email address of the Communicator administrator. All failed emails will be returned to this email address.			

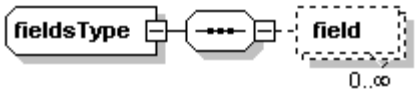
emailSettingsType/subject element

Diagram	
Description	The email subject used when distributing letters through the email distribution channel.

emailSettingsType/body element

Diagram	
Description	The body of the e-mail used when distributing letters through the e-mail distribution channel.

fieldsType complexType

Diagram	
Used by elements	xrm-jar/correspondence/distribution/distribution-templates/distribution-template/distribution-channels/distribution-channel/fields xrm-jar/correspondence/distribution/distribution-templates/distribution-template/fields

fieldsType/field element

Diagram				
Attributes	**Name**	**Type**	**Use**	**Description**
	name	nonEmptyString	required	The field's name. This name may be used in a Cover Page Calligo model.
	pattern	nonEmptyString	optional	A regular expression used to validate the field. See the detailed description below.
Description	Distribution template field			

Pattern Attribute

You can use the `pattern` attribute to validate the data entered into a field, to ensure that it is entered correctly. For example, you can ensure that only digits are entered into a phone or fax number.

The `pattern` attribute is optional. If it is omitted, then any value entered is considered correct.

If the entered value does not match the pattern, then the user will see the following message on the Communicator **Distribution** screen:

> The value entered in the *[name]* field does not match the format specified in the configuration settings.

where *[name]* is the field name.

Note that the message does not specify what the exact error of the entered value. The end users still must know what values are valid for specific fields.

Detailed documentation on patterns and regular expression is at:

```
http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/
Pattern.html
```

Also, the following website contains documentation, tutorials, and examples:

```
http://www.regular-expressions.info/
```

Examples

Here are some examples of regular expression for a fax number field:

Example 1

```
1[\-\.]?(\d[\-\.]?){10}
```

This regular expression reads as follows:

- a ‘1’, followed by
- zero or one dash or a period, followed by
- exactly ten digits

Each digit may be followed by zero or one '-' or '.'. This may be considered somewhat a liberal validation, but it is still a valid fax number.

This pattern will allow these numbers:

- 14165131400
- 1-416-513-1400
- 1-4-1.6-5.1.3.14-00-

It will not allow these numbers:

- 1(416)5131400
- 1 416 513 1400

Example 2

This example contains greater limits on what values can be entered. It forces users to enter the number in the more widely accepted format in North America.

```
1\-\d{3}\-\d{3}\-\d{4}
```

This regular expression can be read as follows:

- a ‘1’, followed by
- one dash, followed by
- exactly 3 digits, followed by
- one dash, followed by
- exactly 3 digits, followed by

- one dash, followed by
- exactly 4 digits

This pattern will allow the following number:

- 1-416-513-1400

It will not allow these numbers:

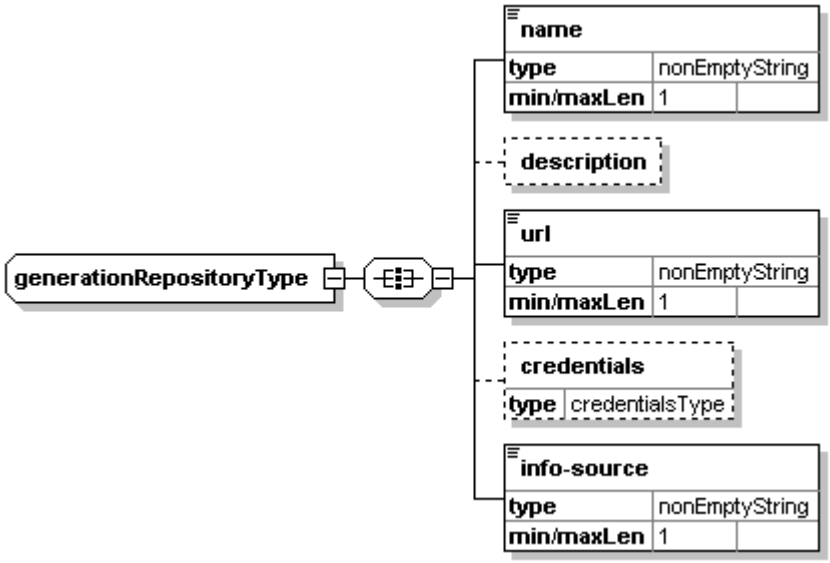
- 1(416)5131400
- 1416-513-1400
- 1416-513-14-00

CorrConfig.xml Example

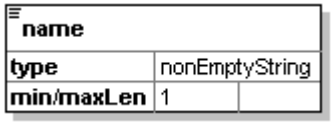
The following code is an excerpt from a sample `CorrConfig.xml` file using the `pattern` attribute:

```
<distribution-template default="false" name="Secure Template">

  <distribution-channels>

    <distribution-channel secure="false" type="email">
      <fields>
        <field name="email"/>
      </fields>
    </distribution-channel>

    <distribution-channel secure="false" type="fax">
      <fields>
        <field name="fax" pattern="1\-\d{3}\-\d{3}\-\d{4}"/>
      </fields>
    </distribution-channel>

    <distribution-channel secure="true" type="print">
      <fields>
        <field name="addr1"/>
        <field name="addr2"/>
        <field name="city"/>
        <field name="zip" pattern="\d{5}(\-\d{4})?" />
        <field name="state"/>
      </fields>
    </distribution-channel>

  </distribution-channels>

  <fields>
    <field name="PersonName"/>
  </fields>
  <description>Default distribution template</description>
</distribution-template>
```

generationRepositoryType complexType

Diagram	
Used by elements	xrm-jar/correspondence/repositories/generation-repositories/auxiliary-repository xrm-jar/correspondence/repositories/generation-repositories/general-repository xrm-jar/correspondence/repositories/generation-repositories/section-repositories/section-repository

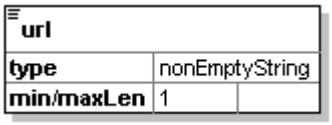
generationRepositoryType/name element

Diagram	
Type	nonEmptyString
Description	The Repository name presented in the Communicator user interface.

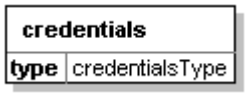
generationRepositoryType/description element

Diagram	
Description	Repository description

generationRepositoryType/url element

Diagram	
Type	nonEmptyString
Description	A URL Communicator uses and Publisher to access the repository.

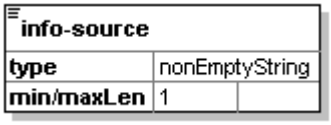
generationRepositoryType/credentials element

Diagram	
Type	credentialsType

Attributes	**Name**	**Type**	**Use**
	username	xs:string	required
	password	xs:string	optional

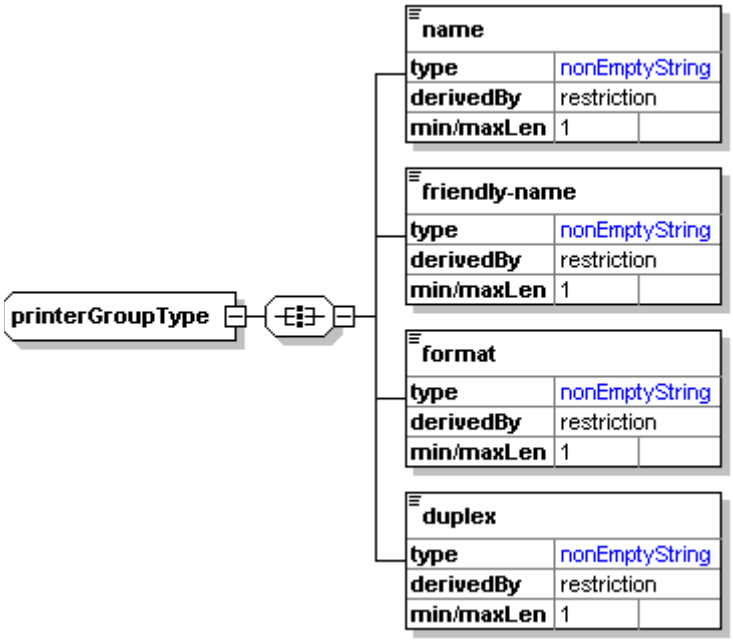
Description	Credentials Communicator uses to access the repository.

generationRepositoryType/info-source element

Diagram	
Type	nonEmptyString
Description	The InfoSource the Calligo Assembly Engine uses to access the repository.

printerGroupType complexType

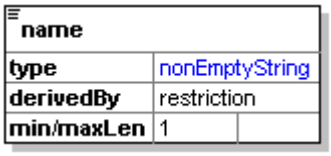
Diagram



Used by elements	xrm-jar/correspondence/printer-groups/default-printer-group xrm-jar/correspondence/printer-groups/printer-group			
Attributes	**Name**	**Type**	**Use**	**Description**
	showForOverride	xs:boolean	Optional	Controls whether to show this printer in the list of printers available for printer override.

printerGroupType/name element

Diagram



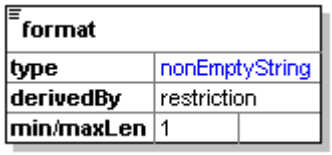
Type	restriction of nonEmptyString
Description	The name of the printer. This name should be available on all Publisher Workers.

Important: Ensure that the printer is correctly specified. If you delete or rename a printer in a printer group, the default printer will be used.

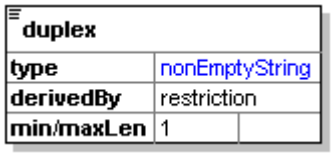
printerGroupType/friendly-name element

Diagram	
Type	restriction of nonEmptyString
Description	The name displayed in the Communicator UI.

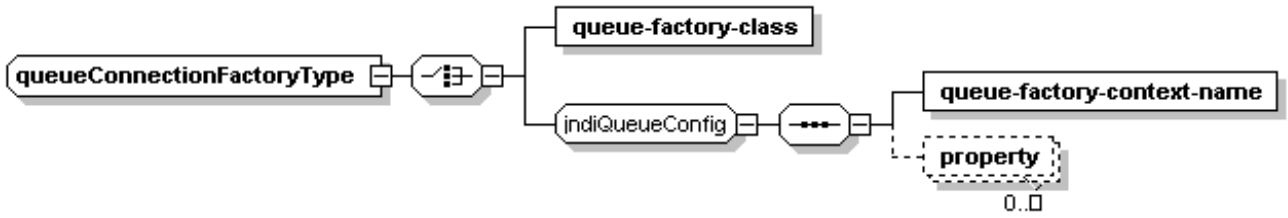
printerGroupType/format element

Diagram	
Type	restriction of nonEmptyString
Description	Specifies the printer stream format. Allowed values are: PCL, PS, Postscript or any mime type.

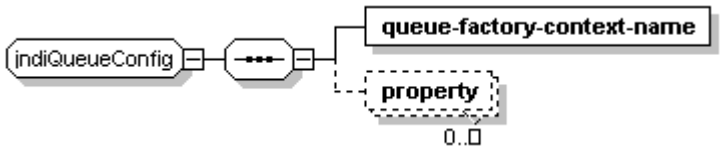
printerGroupType/duplex element

Diagram	
Type	restriction of nonEmptyString
Facets	enumeration none enumeration flipShortEdge enumeration flipLongEdge
Description	Duplex type

queueConnectionFactoryType complexType

Diagram	
Used by elements	corrQueueType/queue-connection-factory docServicesQueueType/queue-connection-factory
Description	queue-factory-class is for MQ series server only. queue-factory-context-name is for OpenJMS only.

jndiQueueConfig group

Diagram	
Used by	queueConnectionFactoryType complex type
Description	JMS Queue Connection parameters using JNDI. For OpenJMS only.

jndiQueueConfig/queue-factory-context-name element

Diagram				
Properties	**isRef**: 0 **content**: complex			
Attributes	**Name**	**Type**	**Use**	**Description**
	contextName	nonEmptyString	required	The JNDI context name of queue connection. For OpenJMS only.

jndiQueueConfig/property element

Diagram				
Properties	**isRef**: 0 **content**: complex			
Attributes	**Name**	**Type**	**Use**	**Description**
	key	nonEmptyString	required	Property name
	value	xs:string	required	Property name
Description	Specifies additional properties to connect to the JNDI provider. For OpenJMS only.			

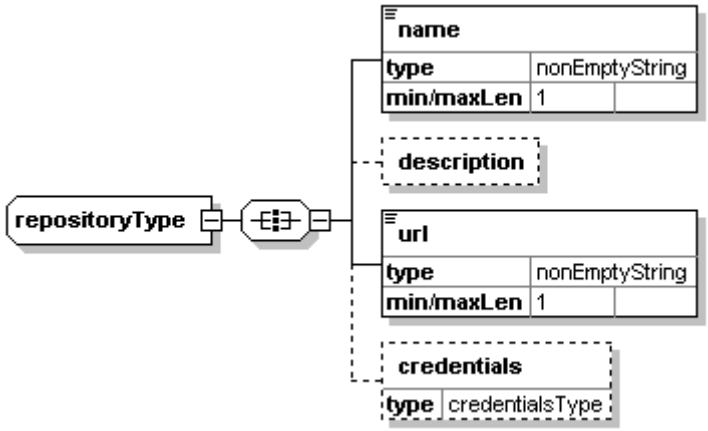
queueConnectionFactoryType/queue-factory-class element

Diagram				
Attributes	**Name**	**Type**	**Use**	**Description**
	class	xs:string	required	This internal configuration is updated by the configuration wizard. Do not edit this manually.
	queueManager	xs:string	optional	MQ Series Server Queue Manager name
	hostName	xs:string	optional	MQ Series Server Host name
	channel	xs:string	optional	MQ Series Server Communication Channel name
	port	xs:short	optional	MQ Series Server Communication Channel port number
	acknowledge	xs:string	optional	This internal configuration is updated by the configuration wizard. Do not edit this manually.
	transportType	xs:string	optional	This internal configuration is updated by the configuration wizard. Do not edit this manually.
Description	MQ Series Server Connection parameters			

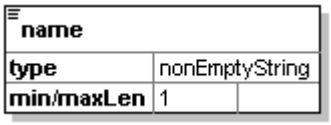
queueElementType complexType

Diagram				
Used by elements	corrQueueType/queue docServicesQueueType/queue			
Attributes	**Name**	**Type**	**Use**	**Description**
	type	xs:string	required	Specifies whether the “name” attribute specifies the JMS Server Queue Name or JNDI context name.
	name	xs:string	required	JMS Server Queue Name or JNDI context name. See “type” attribute.
	user	xs:string	optional	Username for connection to the JMS Server Queue.
	password	xs:string	optional	Password for connection to the JMS Server Queue.
Description	JMS Server Queue parameters			

repositoryType complexType

Diagram	
Used by elements	xrm-jar/correspondence/repositories/kb-repositories/kb-repository xrm-jar/correspondence/repositories/retention-repositories/retention-repository

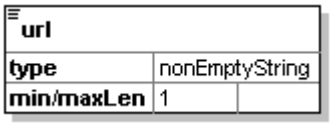
repositoryType/name element

Diagram	
Type	nonEmptyString
Description	The Repository name in the Communicator user interface.

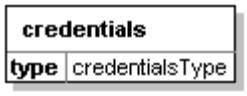
repositoryType/description element

Diagram	
Description	Repository description

repositoryType/url element

Diagram	
Type	nonEmptyString
Description	A URL Communicator and Publisher use to access the repository.

repositoryType/credentials element

Diagram			
Type	credentialsType		
Attributes	**Name**	**Type**	**Use**
	username	xs:string	required
	password	xs:string	optional
Description	The credentials Communicator uses to access the repository.		

Chapter 5

Letter Utility

This chapter describes how to use and configure the Letter Utility. This tool allows you to copy archived letters into the correct location in a DMS and apply the appropriate categories and attribute values.

This chapter includes the following topics:

Note: For installation procedures, refer to *Letter Utility* on page 53 of the *Communicator Installation Guide*.

Overview

IStream Communicator includes functionality that allows letters to be retained in the Calligo Enterprise Document Management System (DMS) for a prolonged period of time. Generated letters can be retained in Microsoft Word, Adobe PDF or TIFF format. By using the DMS, you can retain all of your generated letters in a single, easy to access location. Searching for retained letters in the Calligo DMS using the letter's metadata (its category and attributes) ensures that the retained letters can be quickly accessed.

Letters generated in Communicator are saved in a temporary repository either on a file system or an FTP location. The metadata associated with the letter, along with the variable letter data, is stored in the Communicator database.

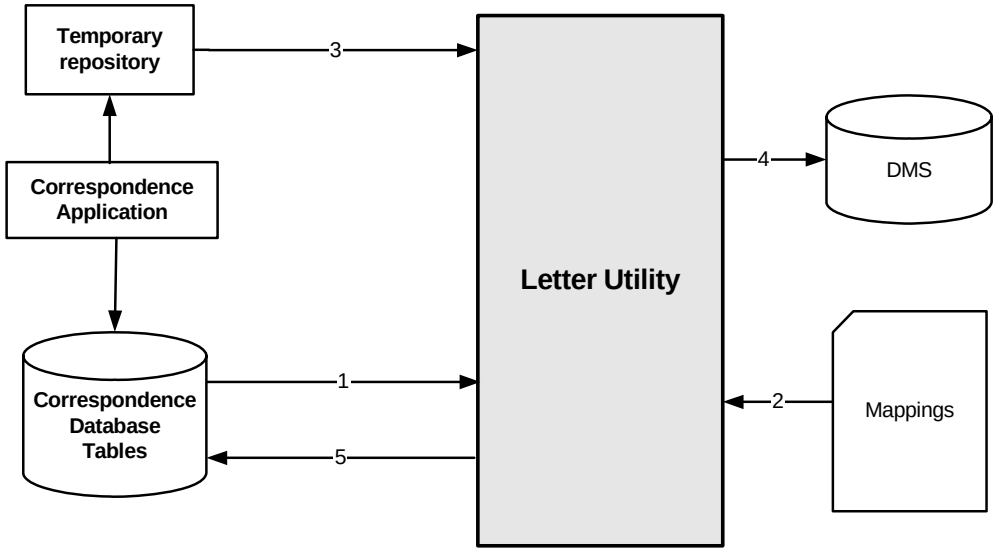
The Letter Utility is used to upload the letters from the temporary repository to the specified location in the Calligo DMS and assigns the applicable metadata to the stored letter.

The Letter Utility logs all of the transactions that it performs for easy review. This logging allows users to trace back a copy of a retained letter using its ID or temporary storage location.

The Letter Utility is usually run daily as an overnight batch process. However, it can also be run from a command prompt if specific processing is required.

Process Overview

The following diagram gives an overview of the letter utility processes:



1. The letter utility retrieves the data associated with the retained letter.
2. The letter utility uses mappings to resolve the category name, attributes and repository URLs for the document and uses this information to prepare a task definition.

3. The letter utility retrieves the retained letter from the temporary location where Communicator has stored it.
4. The letter utility uploads the letter into the Calligo DMS. The object in the DMS is attached object metadata: the category and attribute.
5. The utility's state table is updated with the state of the retained letter.
6. The copy of the letter in the temporary FTP location is deleted if the `policy` parameter in the configuration file is set to `"on"`.

Mapping the Letter Utility

This section describes how the letter utility maps to Communicator.

The letter utility retrieves information from the Communicator database to define tasks. These tasks can be transferring retained letters or deleting files.

For letter transfers, the utility needs to determine the following information for a retained letter:

- the path of the source file: `src_path`
- path of the destination folder: `dest_path`
- object metadata name and value: `om`

To obtain this information from the Communicator database, the utility queries all LIDs (the IDs for the retained letters) which match the following condition:

- `CORRLETTER:STATECODE = '14' and`
- `UTILITY_STATE:STATE != "Archived" and`
- `UTILITY_STATE:FAILURE_COUNTER < max_failures`

where `'14'` indicates `"Archived"` and `max_failures` is a parameter that the utility retrieves from its configuration file, or which can be modified in the GUI

Note: Ensure that there is a retention InfoSource configured in `CorrConfig.xml` for archived letters that have been moved to the DMS.

Mapping Process

The following steps describe the process by which the letter utility obtains the required information:

1. The utility retrieves the following information from its configuration:

- `Category_name` – a string identifying the category of the attributes to associate with the retained letter in the DMS
- `Dest_rel_path` – a string identifying the folder in the DMS in which all the retained letters must be stored (in subfolders of this folder)
- `Folder_def` – a string containing instructions for the letter utility describing how to define the folder for a specific letter definition
- `File_def` – a string containing instructons for the letter utility describing how to define the destination file name

`Folder_def` and `File_def` have the following format:

- `Definition : (Token)+`
- `Token : ${Variable_name} | Text`
- `Text : (character)+`

where `Variable_name` must match the `CORRVARDATA:VARNAME` of the retained letter.

2. The letter utility uses `Category_name` to select the set of attributes that are to be persisted with the retained letter. The letter utility retains a record set of names, values and types of variables from `CORRVARDATA` that match the names of the attributes to use within the definition processing (For more information, see the previous descriptions of `Folder_def` and `File_def`.)

3. Using `CORRLETTER:REPOSITORY`, the letter utility selects a URL from its configuration cache (from the repository settings in `CorrConfig.xml`), and then derives the source path as follows:

```
Source_path = <REPOSITORY_URL> +'/'+ CORRLETTER:REPLOCATION
```

4. The letter utility selects `Dest_rel_path`. After processing `Folder_def` to obtain the folder and `File_def` to obtain the file, it derives the destination path as:

   ```
   Dest_path = Destination_root +'/'+ folder +'/'+ file
   ```

   Note that `Destination_root` is selected with an argument passed to the utility during startup. The letter utility can be used in either a test or production scenario, therefore you must be able to select the destination.

5. The utility letter derives the task definition as:

   ```
   (Source_path, Dest_path, ({Category_name_i.Attr_name,
   Attr_value}*))
   ```

Mapping Example

This section gives an example of mapping the letter utility to Communicator.

Folder Structure

In this example, the folder structure in the DMS is:

1. PRODUCTION
   2. Communicator
      3. Completed Letters
         4. Brokers
            5. Broker #'s delimited by 250 so 000000001 - 000000250
               000000251 - 000000500, etc.
               6. 000000288
                  7. Letter #422003-05-13

         4. Groups
            5. Group #'s delimited by 250 so 0000001 - 0000250
            0000251 - 000500, etc.
               6. 0000488
                  7. Group Letters
                     8. Letter #182003-01-03

8. Letter #32 2003-06-10

7. Member Letters

8. Member 123456789

9. Letter #133 2003-03-02

4. Providers

5. Provider #'s delimited by 250 so 000000001 - 0000000250

000000251 - 000000500, etc.

6. 000000399

7. Letter #159 2003-06-03

3. Component Definitions

3. Letter Definitions

Categories and Attributes

The categories and attributes for this example are:

Attribute Description	Category	Attribute Name (Data Dictionary Variable Name)
Date letter was sent to recipient	Group/Member, Agent, Provider	miscLetReqDt
Employee First Name	Group/Member	sub1stNm
Employee Last Name	Group/Member	subLstNm
Member Number	Group/Member	mbrNbr
Division Name	Group/Member	divName
Division Number	Group/Member	divNum
Group Name	Group/Member	divNameGrp
Group Number	Group/Member	divNbrGrp
Agent/Broker First Name	Agent/Broker	agt1stNm
Agent/Broker Last Name	Agent/Broker	agtLstNm
Agent/Broker Number	Agent/Broker	agtSysAssgnNbr
Provider Number	Provider	prvdrSysAssgnNbr
Provider First Name	Provider	prvdr1stNm
Provider Last Name	Provider	prvdrLstNm
Senders First Name	Group/Member, Agent, Provider	sctyLoggedAssc1stNm
Senders Last Name	Group/Member, Agent, Provider	sctyLoggedAsscLstNm

Configuration Values

The values to add to the configuration in this example are:

- `Dest_Name = DmsRetention` – a repository with this name must be in `Corrconfig.xml`.
- `Dest_root = /Production/Communicator`
- `Dest_rel_path = /CompleteLetters`
- `Source_path` = dynamically obtained from the `REPLOCATION` columns in the `CORRLETTER:REPOSITORY` table
- `Folder_def = ${LETTER_DEF_NAME}/${agtSysAssgnNbr}/ ${mbrNbr}`
- `File_def = Letter #${LETTER_ID} ${miscLetReqDt}`

See *System Attributes* on page 126 for system variables that are available as attributes, and see also the `mapping.xml` file.

Configuration Files

The configuration files for the Letter Utility are:

- `mapping.xml` – contains the category mappings, and must be regularly maintained and updated according to your business needs
- `utilconfig.xml` – contains various configuration parameters for the Letter Utilty functions: descriptions of these parameters are included in this file

Table Logging Database Schema

The utility logs all successful processing into its own table, which is added to the Communicator schema. This table is used for auditing and selecting new tasks. The utility also produces logging messages but these are generally used for debugging.

A `UTILITY_STATE` database table contains the following columns:

- `LID` – the letter identifier
- `SOURCEREPO` – the source repository name
- `SOURCE` – the URL of the file in the temporary repository (from where the letter has been moved)
- `DESTREPO` – the destination repository name
- `DESTINATION` – the URL to the retained letter object in the destination repository
- `ACTDATE` – the date when the object (the retained letter and its attributes) in the destination had been created
- `STATE` – the state of the letter: either `Archived` or `Failed`
- `FAILURE_COUNTER` – the number of times that processing of the retained letter has failed. The letter utility increments this value. This allows the letter utility to reprocess the letter at a later date, but no more than the number of times specified in `max_rl_failure` in the configuration DTD.

To update the Communicator table that tracks the retained letters (`CORRLETTER:REPOSITORY`), the letter utility database schema contains a trigger that refreshes that field from `UTILITY_STATE:DESTINATION` whenever the `UTILITY_STATE:STATE` becomes `'14'` (archived).

System Attributes

The following system attributes are available in the cached data dictionary when running the utility:

- SYSTEM_DATE – the system date, DDMMYYYY format
- SYSTEM_TIME – system time: HHMMSS, for example 183543
- SYSTEM_TIME_MS – returns the number of milliseconds since January 1, 1970, 00:00:00 GMT
- LETTER_DEF_NAME – the letter definition name of the current letter being processed
- SOURCE_FILE_NAME – the original file name from the source retention repository of the current letter being processed
- LETTER_LID – the letter ID of the current letter being processed

Letter Utility Limitations

- The category attribute `default_value` is only supported for string data types.
- `FolderDef` and `fileDef` can only include variables from the Data Dictionary. They cannot include variables that are defined only in mapping.xml.
- According to the RFC2396 standard, the following characters are not supported for the file and folder names in `mapping.xml`:

  `< > # % < >`

  We also recommend that you do not use the following characters:

  `{ } | \ ^ [ ] ``

  This includes Data Dictionary values if Data Dictionary variables are used in the `folderDef` and `fileDef` attributes of category mappings that are defined in `mapping.xml`.
- The Letter Utility does not support a period in the category attribute name, which is the `actual_name` as defined in the mapping XML file. Variable names also cannot contain a period.
- The `source_age` configuration attribute is not supported. If you remove `source policy="on"`, then the source file will be removed immediately after a successful copy of the letter.
- The utility does not support any parameters as command line arguments. Therefore, you cannot pass the `Destination_root` attribute in the configuration as an argument to the utility at startup.
- After archiving letters and then processing them with the Letter Utility, you will no longer be able to view them in the Tracking screen.
- If after deriving the `Dest_path`, the URL points to an existing file in the DMS, the letter utility adds a new version to that object and logs a warning message.
- If you cannot connect to the temporary FTP location or DMS, the utility will fail no more than a maximum number of successive tasks before shutting down, as determined by the `max_successive_failures` in the configuration DTD. If the connection to the database fails, the letter utility shuts down automatically after writing an error log message
- The utility does not support multiple value variables for any retained letter.

Appendix A

API Error Codes

This appendix provides detailed error codes that you may come across while using the SDK or IDM.

Error Code	Description
ACCESS_DENIED_TO_LETTER_ DEFINITION	User with credentials specified in the XML message has no rights to access the Letter Definition.
ADDIN_UNKNOWN	Generation request references an Add-In name unknown to Communicator.
ADDIN_VALIDATION_FAILURE	Add-In validation failed. The problem may relate to the number of copies, required add-ins or a non-specified location for dynamic add-in.
COMPONENT_UNKNOWN	Generation request references a Component name unknown to Communicator.
DISTRIBUTION_TARGET_NOT_ AVAILABLE	Generation request references a distribution target type not supported by Communicator.
DISTRIBUTION_TARGET_NOT_DEF INED	Required distribution target is not defined in the generation request.
DISTRIBUTION_TARGET_ PARAMETERS_NOTDEFINED	Distribution target parameters are not defined in the generation request.
INTERNAL_COMMUNICATION_ ERROR	Communicator internal error related to communication problem.
INTERNAL_CONFIGURATION_ ERROR	Communicator internal error related to invalid configuration.
KEYDATA_NOT_DEFINED	Required key data is not defined in the generation request.
KEYDATA_UNKNOWN	Generation request references a key data name unknown to Communicator.

Error Code	Description
LETTER_DEFINITION_NOT_FOUND	Generation request references a letter definition unknown to Communicator.
RECIPIENT_UNKNOWN	Generation request references a recipient name unknown to Communicator.
RECIPIENT_VALIDATION_ERROR	Recipient data is invalid.
UNKNOWN_ERROR	Unknown internal error.
VARIABLE_DATA_NOT_DEFINED	Required user defined variable data is not defined in the generation request.
VARIABLE_DATA_UNKNOWN	Generation request references a variable data name unknown to Communicator.

INDEX

S

T

U

V

X