Kofax Communications Manager

DID Developer's Guide

Version: 5.4.0

Date: 2020-08-26

KOFAX

Table of Contents

Preface

This guide provides information on how to develop Database Interface Definitions (DIDs) to retrieve data from a database and send it to Kofax Communications Manager (KCM).

This method is considered an alternative, and we recommend that you use the Data Backbone XML file method to send data to KCM instead.

Related documentation

The documentation set for Kofax Communications Manager is available here:[1]

https://docshield.kofax.com/Portal/Products/KCM/5.4.0-cli2a1c07m/KCM.htm

In addition to this guide, the documentation set includes the following items:

Kofax Communications Manager Release Notes
Contains late-breaking details and other information that is not available in your other Kofax Communications Manager documentation.

Kofax Communications Manager Technical Specifications
Provides information on supported operating system and other system requirement for Kofax Communications Manager.

Kofax Communications Manager Installation Guide
Contains instructions on installing and configuring Kofax Communications Manager and its components.

Kofax Communications Manager Getting Started Guide
Describes how to use Contract Manager to manage instances of Kofax Communications Manager.

Kofax Communications Manager Batch & Output Management Getting Started Guide
Describes how to start working with Batch & Output Management.

Kofax Communications Manager Repository Administrator's Guide
Describes administrative and management tasks in Kofax Communications Manager Repository and Kofax Communications Manager Designer for Windows.

[1] You must be connected to the Internet to access the full documentation set online. For access without an Internet connection, see "Offline documentation" in the Installation Guide.

Kofax Communications Manager Repository User's Guide
Includes user instructions for Kofax Communications Manager Repository and Kofax Communications Manager Designer for Windows.

Help for Kofax Communications Manager Designer
Contains general information and instructions on using Kofax Communications Manager Designer, which is an authoring tool and content management system for Kofax Communications Manager.

Kofax Communications Manager Template Scripting Language Developer's Guide
Describes the KCM Template Script used in Master Templates.

Kofax Communications Manager Core Developer's Guide
Provides a general overview and integration information for Kofax Communications Manager Core.

Kofax Communications Manager Core Scripting Language Developer's Guide
Describes the KCM Core Script.

Kofax Communications Manager Batch & Output Management Developer's Guide
Describes the Batch & Output Management scripting language used in KCM Studio related scripts.

Kofax Communications Manager Repository Developer's Guide
Describes various features and APIs to integrate with Kofax Communications Manager Repository and Kofax Communications Manager Designer for Windows.

Kofax Communications Manager ComposerUI for ASP.NET Developer's Guide
Describes the structure and configuration of KCM ComposerUI for ASP.NET.

Kofax Communications Manager ComposerUI for J2EE Developer's Guide
Describes JSP pages and lists custom tugs defined by KCM ComposerUI for J2EE.

Kofax Communications Manager ComposerUI for ASP.NET and J2EE Customization Guide
Describes the customization options for KCM ComposerUI for ASP.NET and J2EE.

Kofax Communications Manager API Guide
Describes Contract Manager, which is the main entry point to Kofax Communications Manager.

Getting help with Kofax products

The Kofax Knowledge Base repository contains articles that are updated on a regular basis to keep you informed about Kofax products. We encourage you to use the Knowledge Base to obtain answers to your product questions.

To access the Kofax Knowledge Base, go to the Kofax website and select **Support** on the home page.

Note The Kofax Knowledge Base is optimized for use with Google Chrome, Mozilla Firefox or Microsoft Edge.

The Kofax Knowledge Base provides:

- Powerful search capabilities to help you quickly locate the information you need.

  Type your search terms or phrase into the **Search** box, and then click the search icon.
- Product information, configuration details and documentation, including release news.

  Scroll through the Kofax Knowledge Base home page to locate a product family. Then click a product family name to view a list of related articles. Please note that some product families require a valid Kofax Portal login to view related articles.
- Access to the Kofax Customer Portal (for eligible customers).

  Click the **Customer Support** link at the top of the page, and then click **Log in to the Customer Portal**.
- Access to the Kofax Partner Portal (for eligible partners).

  Click the **Partner Support** link at the top of the page, and then click **Log in to the Partner Portal**.
- Access to Kofax support commitments, lifecycle policies, electronic fulfillment details, and self-service tools.

  Scroll to the **General Support** section, click **Support Details**, and then select the appropriate tab.

Chapter 1

Introduction to Database Interface Definition

Documents created with KCM can use data from a data source, which typically is a Data Backbone XML file that is sent to KCM to create documents. In cases where KCM needs to retrieve the data *directly* from a data source, you can use a DID, which is a description of a database. When a Master Template created with a DID is running, KCM retrieves the data from the data source.

DIDs are created in KCM Designer for Windows using the DID specification language.

A DID is a map that contains database files, external data retrieval programs, XML tags, and SQL queries. A DID is created from a **DID document** that has two levels:

1. **DID module level (or DID source level):** On this level, you define DID components such as entries and functions.
2. **DID definition level:** On this level, you declare which of these components on the module/source level are accessible in Master Templates.

You can create DID modules for different connection types and combine these DID modules in a single DID (see Configure connection types).

About DID elements

This section describes the various elements that constitute a DID.

A DID structure is composed of entries (files and folders), subentries, and functions.

An **entry** defines what data is retrieved and enables you to retrieve the data from a data source. An entry definition consists of Fields, a data retrieval method, and a set of formal parameters. **Formal parameters** are required when the DID entry is used in a Master Template with the `PATH/PAR` construction. Also, you can define an entry as a **main entry** to use as a starting point for the data retrieval in a Master Template.

The data that an entry retrieves can conform to a database record, such as a table in an SQL database. It may not coincide with an actual table in the database, but can present a pre-processed, simplified or summarized view of a single table or a collection of tables. For information on the syntax of an entry, see Entries.

An entry can have a number of **subentries**. Also, you can export entries defined in a DID module and import them as subentries in another DID module (see Subentries). A subentry definition consists of **actual parameters**. The actual parameters are the values of the formal parameters used to access a particular subentry of an entry.

A data set retrieved by an entry is contained in a **Field**, and the retrieved data can only be accessed through these Fields. Each Field has a data type: In the KCM Template scripting language, numerical

Fields are mapped to the `NUMBER` type, while alphanumerical Fields are mapped to the `TEXT` type. For information on the syntax of a Field, see Fields.

To perform actions on a host system with a DID, use **DID defined functions**. For example, you can calculate totals, subtotals or averages, or update records in a database. For information on the syntax of the DID functions, see DID defined functions.

Additionally, an entry describes the input and output parameters for the **data retrieval** method, which is the core of the data-text merge process. Depending on the connection type, one or more mechanisms are available for data retrieval. The result of data retrieval is a data set that conforms to the Field definitions of the entry.

A data retrieval method has a variant called **key retrieval**. It determines the Fields to show in the Key Selection window when the Master Template is run and the actual parameters to pass to the data retrieval method.

Chapter 2

Work with DIDs

This chapter describes how to create and edit a DID using the DID specification language.

DIDs are created and stored in KCM Designer for Windows. When you create a new project, the DIDs folder is automatically created for that project.

The DIDs folder also contains the Includes folder used for modular DIDs. A modular DID is a DID that contains the `__INC(filename)` instruction that you can use to include a file in a DID. This file is stored in the Includes folder. You can use modular DID to share parts of a DID with another DID.

Create and edit a DID

This procedure shows how to create a DID from a DID document.

Note To create a DID from a DID document, you need a DID development (SDK/MP) license.

1. Start **KCM Designer for Windows**.
2. In the project tree view, click the **DIDs** folder.
3. On the menu, click **File** > **New** > **New DID**.
4. In the **New DID** window, enter the following information:
   - In the **DID Name** field, enter the name. It is limited to 10 characters.
   - In the **Identification (three-letter-code)** field, enter a code to identify the DID during the Master Template development. The code must only contain capital letters. By default, it is set to KCM.
   - In the **DID Module platform** list, select a platform. The platform defines the connection type used to connect to a database.
5. Click **OK**.
   Microsoft Word opens a new DID document, containing all required attributes for the DID (DID definition) and an empty DID module.
   This example shows an empty DID document for the DID named `TestDID` with the ODBC connection type selected.

```
DEFINE_DID
   IDENTIFICATION ITP

   DIDMODULE_LIST
      "TestDID"


   MAIN_ENTRY_LIST
      (* Add main entries to this list *)
```

```
   DID_DEFINED_FUNCTION_LIST
      (* Add DID defined functions to this list *)


END_DEFINE_DID (* TestDID *)


DEFINE_DIDMODULE
   NAME           "TestDID"
   CONNECTION     ODBC

   (* Insert the DID entry definitions here *)

END_DEFINE_DIDMODULE (* TestDID *)
```

6. Make the necessary changes to the DID document using the specification language. For information on the DID document syntax and examples, see DID document syntax. For information on the connection types, see Configure connection types.
   - Using the DID Wizards, you can generate an entry definition, depending on the database type. For more information, see Generate an ODBC or Oracle entry and Generate an ODBC stored procedure entry.
7. On the **ITP/MDK** tab, click **Create DID**.

   The DID is created for the DID document.
8. Save the changes and close the DID document.
   - To edit or view the DID document, right-click it and click **Edit** or **View**, respectively. These actions are only available for DIDs created in KCM Designer for Windows.

Note If a DID document is included in another DID, do not change its name.

Generate an ODBC or Oracle entry

Using the DID Wizards, you can generate an entry definition based on an ODBC or Oracle database or an ODBC stored procedure. Then you can include the new entry in your DID document.

1. On the **ITP/MDK** tab, in the **DID Wizards** section, click **New ODBC Entry** or **New Oracle Entry**, respectively.

   The wizard appears.
2. Click **Next**, select a driver from the list, and then click **Next** again.
3. Connect to the server and click **OK**.
4. In the subsequent dialogs, make selections where needed and click **Next**.
   - If required, restrict the list of tables to retrieve from the database.
   - In the list of tables, select a database table to include in the new entry.
   - In the list of Fields, select the Fields to retrieve.
   - Choose formal parameters for the new entry to call it from a Master Template.
   - Enter a name for the new entry and select a type. Select **WITH** for a single entry to retrieve one record or **FORALL** for an entry where zero, one, or more records need to be retrieved.

5. When you receive a notification that the new entry is created, click **Finish** to insert it in your DID document.
   The new entry is inserted. Make changes if needed and save the document.

Generate an ODBC stored procedure entry

1. On the **ITP/MDK** tab, in the **DID Wizards** section, click **New Stored Procedure Entry**.
   The wizard appears.
2. Click **Next**, select a driver from the list, and then click **Next** again.
3. Connect to the server and click **OK**.
4. In the subsequent dialogs, make selections where needed and click **Next**.
   - If required, restrict the list of tables to retrieve from the database.
   - In the procedures list, select a procedure to include in the new entry.
   - Enter a name for the new entry and select a type. Select **WITH** for a single entry to retrieve one record or **FORALL** for an entry where zero, one, or more records need to be retrieved.
5. When you receive a notification that the new entry is created, click **Finish** to insert it in your DID document.
   The new entry is inserted. Make changes if needed and save the document.

Upload a DID document to a project

You can upload an existing DID document to KCM Designer for Windows from your computer.

1. Right-click the **DIDs** folder to upload a new DID document and click **Load DID document**.
2. Select the file on your computer and click **Open**.
   The following formats are supported: TXT, DOC, DOCX.
   The new DID document is added to the system.
   a. To create a DID from this DID document, right-click it and click **Edit**.
      Microsoft Word is opened.
   b. On the **ITP/MDK** tab, click **Create DID**.
      The new DID is created and added to the DIDs folder.

Upload an existing DID to a project

You can upload an existing KCM DID from your computer to KCM Designer for Windows.

1. Right-click the **DIDs** folder to upload an existing KCM DID file and click **Load DID**.
2. Select the file on your computer and click **Open**.
   The DID is added to the system.

Use a DID wizard to create a DID document

You can generate a DID document from an XML document containing the required data to create a DID.

1. Right-click the **DIDs** folder to contain a new object and click **Run DID wizard**.
2. In the drop-down list, select a base file type:
   - Select **New XML-File DID document** for an XML file.
   - Select **New XSD-Based XML-FIle DID document** for an XSD file.
   - Select **New WSDL XML-Web DID document** for a WSDL file.
3. Click **OK**, make a selection where needed, and then click **Next** to continue.
4. Specify the name and three-letter code for the new DID document and click **Next**.
5. Click **Finish** when the new DID document is created.
   The document is opened in Microsoft Word.
6. Verify the document and make changes, if required.
7. To create a DID from this DID document, on the **ITP/MDK** tab, click **Create DID**.
   The new DID is added to the DIDs folder.

Browse a DID

After a DID is created, you can view its elements, such as entries, subentries, Fields, and functions, using DID Browser. Also, you can copy some of those elements and use them to generate parts of a Master Template (see Copy DID elements to use in a Master Template).

1. Right-click a DID and click **Browse DID**.
   When the DID is opened in Microsoft Word, click **Browse DID** on the **ITP/MDK** tab.
   The DID Browser appears.
2. The central window shows a list of main entries of the DID. To see all entries, clear the **Main entries only** check box.
   - To get technical details about the entries, select **Technical details**, select an entry, and then click **Details**. You can see the details of any item belonging to the DID.
   - To see the Fields used in an entry, click **Fields**.
   - To view the parameters of an entry, click **Formal par**.
   - To check subentries of an entry, click **Subentries**.
     - In addition to the general set of actions, you can see the parameters of a subentry. Click **Actual par.**
   - To view the standard functions of an entry, click **Stnd. func**.
3. When finished, click **Quit**.

Copy DID elements to use in a Master Template

In DID Browser, you can copy entries, Fields, and functions and use them in a Master Template script. After you copied an element, paste it in your Master Template script.

Open DID Browser, click the needed element, click **Copy**, and then select one of the following options, as applicable.

- **Entries**
  - To copy the script instructions to access an entry, including its subentries, click the entry and click **Copy** > **Copy**.

```
WITH X0 IN EXP.Customer DO

OD (* WITH Customer *)
```

  - To copy the script instructions to access an entry through Template script procedures, click **Copy** > **Copy - procedure**. The procedure includes the `PATH/PAR` expression and the entry formal parameters.

```
PROC customer (CONST NUMBER customer_number)
DO
  WITH X0 PATH EXP.Customer
  WHERE
    PAR(1) = customer_number
  DO

    (* Model text *)

  OD (* WITH X0 *)
OD (* PROCEDURE customer *)
```

- **Subentries**
  - To copy the script instructions to access a subentry, click the subentry and click **Copy** > **Copy**.

```
  WITH X5 IN X0.Address_of_customer DO

  OD (* WITH Address_of_customer *)
```

  - To copy the script instructions to access a subentry, including the path from the main entry to the subentry, click **Copy** > **Copy - entry path**.

```
WITH X0 IN EXP.Customer DO

  WITH X5 IN X0.Address_of_customer DO

  OD (* WITH Address_of_customer *)

OD (* WITH Customer *)
```

  - To copy the script instructions to access a subentry through Template script procedures, click **Copy** > **Copy - procedure**. The procedure includes the `PATH/PAR` expression and the subentry actual parameters.

```
PROC address_of_customer (CONST NUMBER customer_number)
DO
  WITH X5 PATH EXP.Customer.Address_of_customer
  WHERE
    PAR(1) = customer_number
  DO
```

```
    (* Master Template text *)

  OD (* WITH X5 *)
OD (* PROCEDURE address_of_customer *)
```

- **Fields contained in an entry or subentry**
  - To copy the Field contents, click the Field and click **Copy** > **Copy**.

    ```
    #
    Customer_number: @(number( X0.Customer_number ; 0 ))
    #
    ```
  - To copy the Field contents and script instructions to access all entries before the entry/subentry that contains the Field, click **Copy** > **Copy - entry path**.

    ```
    WITH X0 IN EXP.Customer DO
    #
    Customer_number: @(number( X0.Customer_number ; 0 ))
    #
    OD (* WITH Customer *)
    ```
  - To copy the Field contents to use as a variable in a Template script, click **Copy** > **Copy - variables**.

    ```
    NUMBER customer_number := X0.Customer_number
    ```

Upload a DID created on "IBM System i" to a project

You can use a DID created on the "IBM System i" database in KCM Designer for Windows. Usually, these DIDs are created with a KCM component known as ITP SDK AS/400.[2] To use such a DID, you need to connect to the database and upload the respective DID to your project.

1. Right-click the **DIDs** folder and click **Download iSeries DID**.
2. Fill in the fields.
   - In the **DID Name** field, enter the name of the DID to upload.
   - In the **SRC Library** field, enter the SRC library that contains the DID. For example, ITPSRC21 or ITPSRC35.
   - In the **Hostname:Port** field, enter the host name or IP address of the database and optionally the port number that the database is listening to. Separate the two parts with a colon.

     Example

Hostname:Port example	Description
10.0.88.1:10042	The database with IP address 10.0.88.1 is listening to port 10042.
10.0.88.1	The database with IP address 10.0.88.1 is listening to the default port. When the port number is not specified, the default port is used.
myhost:10042	The database with the host name "myhost" is listening to port 10042.
myhost	The database with the host name "myhost" is listening to the default port. When the port number is not specified, the default port is used.
   - In the **User ID** and **Password** fields, enter the ID and password used to log in to the database.
   - In the **Password level** list, select **0** (other values are currently not supported).
3. Click **OK**.

[2] ITP is a former name for KCM; AS/400 is a former name for "IBM System i" databases.

Chapter 3

DID specification language

With the KCM DID specification language, you can define a DID. The language has no executable statements or control structures and follows the same lexical rules as the KCM Template scripting language. The language is free-format: tabs, end-of-lines, end-of-paragraphs, and spaces are treated as white space. You can add comments to the DID document, placing them inside two asterisks.

When using names of KCM identifiers in the KCM DID specification language, begin the name with an uppercase character. It can be followed by lowercase characters, uppercase characters, digits, or underscores. The identifiers defined in the DID specification language are the names of entries and Fields that a script developer uses in a Master Template.

DID document syntax

A DID document consists of a DID definition and a sequence of DID module definitions.

Syntax

```
did-document ::=
did-definition did-module-definition-sequence
```

Example

```
DEFINE_DID
 NAME Example
 (* ... *)
END_DEFINE_DID
DEFINE_DIDMODULE
 NAME Module1
 (* ... *)
END_DEFINE_DIDMODULE
DEFINE_DIDMODULE
 NAME Module2
 (* ... *)
END_DEFINE_DIDMODULE
```

DID definition

In the DID definition, you define the following attributes:

- Name

  Sets the name of the DID. The name can contain a maximum of 10 characters. It must begin with an uppercase character and can be followed by lowercase characters, uppercase characters, digits, or underscores.

  ```
  did_name ::=
  uppercase-itp-name
  ```

```
| quoted-string
```

Note The `did_name` attribute is omitted in a DID created and compiled in KCM Designer for Windows. In KCM Designer for Windows, the DID name is equal to that of the DID object.

- Three-letter code

  Sets the three-letter code that a script developer uses in a Master Template when accessing a main entry.

  ```
  three-letter-code ::=
  uppercase-character
  ```

- DID modules

  Specify the DID modules used in this DID.

  ```
  did-module-name ::=
  uppercase-itp-name
  | quoted-string
  ```

- Main entries

  Specify which entries a script developer can use as a main entry in a Master Template.

  ```
  entry-name ::=
  uppercase-itp-name
  ```

- DID functions

  Specify which DID functions a script developer can use in a Master Template.

  ```
  function-name ::=
  uppercase-itp-name
  ```

Syntax

```
did-definition ::=
DEFINE_DID
NAME did_name
IDENTIFICATION three-letter-code
DID_MODULE_LIST [ did-module-declaration-sequence ]
MAIN_ENTRY_LIST [ main-entry-sequenceuence ]
DID_DEFINED_FUNCTION_LIST [ did-defined-function-sequence ]
END_DEFINE_DID
did-module-declaration-sequence ::=
did-module-name [ did-module-declaration-sequence ]
main-entry-sequenceuence::=
main-entry [ main-entry-sequenceuence ]
main-entry ::=
entry-name FROM did-module-name
did-defined-function-sequence ::=
did-defined-function [ did-defined-function-sequence ]
did-defined-function ::=
function-name FROM did-module-name
```

Note If the `NAME` parameter is omitted in a DID created and compiled within KCM Designer for Windows, the name of the DID is equal to that of the DID object.

Example

```
DEFINE_DID
   NAME Example
   IDENTIFICATION EXP
```

```
   DID_MODULE_LIST
         Example_module
   MAIN_ENTRY_LIST
         Customer FROM Example_module
         Order FROM Example_module
   DID_DEFINED_FUNCTION_LIST
         Open_amount_customer FROM Example_module
END_DEFINE_DID
```

DID module

In the DID module, you define the following attributes:

- Name

  Sets the name of the DID module. This is the name used in the DID document to refer to this DID module. The name must begin with an uppercase character and can be followed by lowercase characters, uppercase characters, digits, or underscores.

  ```
  did-module-name ::=
  uppercase-itp-name
  | quoted-string
  ```

- Connection type

  You can create a DID that connects to different database management systems. KCM supports the following connection types: `LOCAL`, `AS400`, `ODBC`, `ORACLE`, `XMLFILE`, `XMLWEB`, `XML MQSeries`, and `MAINFRAME`.

  ```
  connection-type-name ::=
  AS400
  | LOCAL
  | MAINFRAME
  | ORACLE
  | ODBC
  | XMLFILE
  | XMLWEB
  | MQXML
  ```

  For information on the connection types, see Configure connection types. Also, you can use DID Wizards to create entries for some connection types (see Generate an ODBC or Oracle entry and Generate an ODBC stored procedure entry).

  All entries and functions defined in a DID module use the same connection type. You cannot define an entry for various connections in the same DID module.

- Code page

  Sets the default code page for all entries in the DID module. In an entry definition, you can overwrite the code page for an individual entry.

  ```
  codepage ::=
  EBCDIC
  | ASCII
  | natural-number
  ```

- Namespace mappings

  A DID module can specify mappings for XML namespaces. These mappings are applied when data is retrieved from an XML data source.

  ```
  namespace-list
  namespace-list ::=
  namespace-entry [ namespace-list ]
  ```

- Entry definitions

  A DID module can contain one or more entry definitions. You need to declare which entries can be used as main entries within the DID definition.
- Function definitions

  A DID module can contain one or more function definitions. You need to declare it within the DID definition before it can be used in a Master Template.
- Alias entries

  You can declare aliases for entries. You can use these aliases as subentries in the DID module. You cannot declare an alias as a main entry.
- Import entries

  You can import entries from other DID modules as subentries. You cannot declare an import to be a main entry, but you can make aliases for an imported entry.

Syntax

```
did-module-definition ::=
DEFINE_DIDMODULE
NAME did-module-name
CONNECTION connection-type-name
[ did-module-attribute-sequence ]
[ entry-and-func-definition-sequence ]
END_DEFINE_DIDMODULE
did-module-attribute-sequence ::=
did-module-attribute [ did-module-attribute-sequence ]
did-module-attribute ::=
did-module-codepage
| did-module-namespaces
did-module-codepage ::=
CODEPAGE codepage
did-module-namespaces ::=
NAMESPACE namespace-list
namespace-list ::=
namespace-entry [ namespace-list ]
namespace-entry ::=
"prefix" = "URI"
entry-and-func-definition-sequence ::=
entry-and-func-definition [ entry-and-func-definition-sequence ]
entry-and-func-definition ::=
entry-definition
| did-defined-function-definition
| alias-definition
| import-definition
```

Example

```
DEFINE_DIDMODULE
 NAME Example_module
 CONNECTION_TYPE XMLFILE
NAMESPACE
  ""      = "http://example.com//ns//default" (* Default namespace *)
  "xhtml" = "http://www.w3.org//2017//xhtml"  (* xhtml: prefix *)

 (* ...
 Entry and function definitions are placed here
 ... *)
END_DEFINE_DIDMODULE
```

Included documents

You can use the `_INC(filename)` statement to include documents in your DID. Such documents are called subdocuments. You need to manually add the subdocument to include in the Includes folder located in the DIDs folder. When processing the `DID`, KCM then replaces the `_INC(filename)` instruction with the contents of the `filename` file from the Includes folder.

In your DID document, place the instruction `_INC(filename)` on a single line, after the `DEFINE_DID` statement. KCM ignores all text before `DEFINE_DID`.

The `_INC` instruction cannot contain spaces, line breaks, or formatting instructions (bold, italics or other) between the underscore and the closing bracket. The file name cannot contain letters with special accents, such as ö, é, or ñ.

KCM allows multiple levels of subdocuments. The number and size of included documents are limited by the amount of available memory.

If the process fails to include a subdocument, it continues to search for other subdocuments.

Entries

An entry is a DID element that enables you to define the data to retrieve.

Syntax

```
entry-definition ::=
DEFINE_ENTRY
NAME entry-name
 entry-attribute-sequence
[ entry-parameters ]
entry-fields
[ entry-subentries ]
END_DEFINE_ENTRY
entry-attribute-sequence ::=
model-document-statement
data-retrieval
[ key-retrieval ]
[ local-connection-entry-attributes ]
[ as400-connection-entry-attributes ]
[ database-file ]
[ database-record-id ]
[ entry-codepage ]
[ entry-export ]
model-document-statement ::=
MODEL_DOCUMENT_STATEMENT single-or-plural
single-or-plural ::=
WITH
| FORALL
data-retrieval::=
DATA_RETRIEVAL retrieval-name
key-retrieval ::=
KEY_RETRIEVAL retrieval-name
local-connection-entry-attributes ::=
entry-calling-convention
```

```
entry-calling-convention ::=
CALLING_CONVENTION calling-convention
as400-connection-entry-attributes ::=
entry-record-length
entry-record-length ::=
RECORD_LENGTH natural-number
database-file ::=
DATABASE_FILE file-name
database-record-id ::=
DATABASE_RECORD_ID file-name
entry-codepage ::=
CODEPAGE codepage
entry-export ::=
EXPORT_ENTRY yes-or-no
yes-or-no ::=
YES
| NO
entry-parameters ::=
DEFINE_PARAMETERS formal-parameter-sequence
formal-parameter-sequence END_DEFINE_PARAMETERS
field-sequence
entry-fields ::=
DEFINE_FIELDS field-sequence END_DEFINE_FIELDS
```

Example

```
DEFINE_ENTRY
 NAME Customer
 MODEL_DOCUMENT_STATEMENT WITH
 DATA_RETRIEVAL (* ... *)
 KEY_RETRIEVAL (* ... *)
DEFINE_PARAMETERS
  (* ...
  Parameter definitions are placed here
  ... *)
 END_DEFINE_PARAMETERS
 DEFINE_FIELDS
  (* ...
  Add the Field definitions here
  ... *)
 END_DEFINE_FIELDS
END_DEFINE_ENTRY
```

Entry attributes

All entries have the following attributes:

- Name

  Sets the name of the entry. This name is used in the Master Template development. It must begin with an uppercase character and can be followed by lowercase characters, uppercase characters, digits, and underscores.

  ```
  entry-name ::=
  uppercase-itp-name
  ```

- Master Template statement

  Determines whether the entry is single or plural. It can have the values `WITH` for a single entry to retrieve one record or `FORALL` for an entry where zero, one, or more records need to be retrieved.

  ```
  model-document-statement ::=
  MODEL_DOCUMENT_STATEMENT single-or-plural
  single-or-plural ::=
  ```

```
WITH
| FORALL
```

- Data retrieval method

  Determines how to retrieve the data. Depending on the connection type specified in the DID module, this attribute can be the name or full path to a program, query, or special value. The data retrieval string that you enter here is stored in the DID. The content of the string is only verified during the Master Template execution.

```
data-retrieval::=
DATA_RETRIEVAL retrieval-name
```

- Key retrieval method

  Only relevant for main entries. Determines how to retrieve the data to show in the Key Selection window. Depending on the connection type specified in the DID module, this attribute can be the name or full path to a program, query, or special value. The data retrieval string that you enter here is stored in the DID. The content of the string is verified during the Master Template execution.

```
key-retrieval ::=
KEY_RETRIEVAL retrieval-name
```

- Code page

  Sets the code page for the entry. KCM uses the code page defined in the DID module, if any. This attribute is used to translate the contents of Fields and parameters. During the Master Template execution, KCM determines the code page of the system on which it runs. Parameters passed to data retrieval or key retrieval are translated from the system code page to the code page defined in the entry. Fields yielded by data retrieval or key retrieval are translated from the code page defined in the entry to the code page used by the system.

```
codepage
::=EBCDIC
|         ASCII
|         natural-number
```

- Export

  Optional. Defines if the entry should be exported from the DID module. You can import exported entries into other DID modules and use them as subentries. By default, the attribute is set to `NO`.

```
entry-export ::=
EXPORT_ENTRY yes-or-no
yes-or-no ::=
YES
| NO
```

- List of formal parameters

  Optional. You can define formal parameters for an entry. The usage of the parameters is determined by data retrieval and key retrieval. The formal parameters are mostly used as a set of keys that determine which record or row to retrieve from the database.

```
entry-parameters ::=
DEFINE_PARAMETERS formal-parameter-sequence END_DEFINE_PARAMETERS
```

- List of Fields

  Each entry contains at least one Field. Ensure that the data retrieval method delivers the defined Fields, with the correct data type and in the predefined order.

```
field-sequence
entry-fields ::=
DEFINE_FIELDS field-sequence END_DEFINE_FIELDS
```

- List of subentries
  Optional. You can define a list of entries to be used as subentries.
- Database file
  Used by some connection types. When not used, it is silently ignored.
- Database record ID
  Used by some connection types. When not used, it is silently ignored.

In addition to this set of parameters, an entry can have a calling connection attribute that depends on the connection type specified.

Subentries

Within an entry definition, you can define subentries. Each subentry definition consists of the entry name and the where-statement. The where-statement defines the actual parameters to use for the formal parameters of the subentry.

The actual parameters can be either a constant or a Field of the current entry.

Syntax

```
entry-subentries ::=
DEFINE_SUBENTRIES Subentry-sequence END_DEFINE_SUBENTRIES
Subentry-sequence ::=
Subentry-def [ Subentry-sequence ]
Subentry-def ::=
entry-name where-statement
where-statement ::=
WHERE parameter-overwrite [ parameter-overwrite-sequence ]
parameter-overwrite-sequence ::=
parameter-overwrite [ parameter-overwrite-sequence ]
parameter-overwrite ::=
PAR '(' natural-number  ')' '=' actual-parameter
actual-parameter ::=
field-or-constant
field-or-constant ::=
field-name
| CONST quoted-string
```

Example

```
DEFINE_SUBENTRIES
 Postal_adres_of_customer WHERE
  PAR(1) = Customer_number
  PAR(2) = CONST "POSTAL"
 Orders_of_customer WHERE
  PAR(1) = Customer_number
END_DEFINE_SUBENTRIES
```

DID defined functions

With the DID defined functions, you perform different actions on a host system.

Syntax

```
did-defined-function-definition ::=
DEFINE_DID_FUNCTION
NAME function-name
function-attribute-sequence
[ function-parameters ]
function-result
END_DEFINE_DID_FUNCTION
function-attribute-sequence ::=
execute-function
[ local-connection-function-attributes ]
[ function-codepage ]
execute-function ::=
EXECUTE_FUNCTION retrieval-name
local-connection-function-attributes ::=
function-calling-convention
function-calling-convention ::=
CALLING_CONVENTIONS calling-convention
function-codepage ::=
CODEPAGE codepage
function-parameters ::=
DEFINE_PARAMETERS formal-parameter-sequence
END_DEFINE_PARAMETERS
formal-parameter-sequence ::=
field-sequence
function-result ::=
FUNCTION_RESULT field-def
```

Example

```
DEFINE_DID_FUNCTION
 NAME Total_amount_customer
 EXECUTE_FUNCTION ... (* *)
 DEFINE_PARAMETERS
  ...
  (*Add the parameter definitions here*)
  ...
 END_DEFINE_PARAMETERS
 FUNCTION_RESULT ... (* *)
END_DEFINE_DID_FUNCTION
```

Function attributes

DID defined functions have the following attributes:

- Name

  Sets the name of the function. This name is used in the Master Template development. It must begin with an uppercase character and can be followed by lowercase characters, uppercase characters, digits, or underscores.

  ```
  function-name ::=
  uppercase-itp-name
  ```

- Execute function

  Determines the operation to perform when the Master Template calls a DID defined function. Depending on the connection type specified in the DID module, the name can be the full path to a program, an SQL query, or a special value. The string that you enter here is stored in the DID. The content of the string is only verified during the Master Template execution.

```
execute-function
[ local-connection-function-attributes ]
[ function-codepage ]
execute-function ::=
EXECUTE_FUNCTION retrieval-name
```

- List of formal parameters

  Optional. Specifies formal parameters for a function. The formal parameters are used in the execute function.

```
formal-parameter-sequence ::=
```

- Code page

  Sets the code page for the execute function used to translate the result and parameters. If not defined, the code page defined in the DID module is used. During the Master Template execution, KCM determines the code page of the system on which it runs. Parameters passed to the execute function are translated from the system code page to the code page defined in the entry. The result yielded by the execute function is translated from the code page defined in the entry to the code page used by the system.

```
codepage
::=EBCDIC
|          ASCII
|          natural-number
```

- Function result

  Result of a function that was called.

```
function-result ::=
FUNCTION_RESULT field-def
```

In addition to this set of parameters, a DID defined function can have a calling connection attribute that depends on the connection type specified.

Aliases and imports

Alias

To use a single entry twice with different actual parameters, you can make one or more aliases for this entry. When creating an alias for a subentry, you can only use the entry and the alias once. You cannot declare an alias to be a main entry.

Import

You can import exported entries into other DID modules and use them as subentries as follows:

- You can split your DID according to the subsystems that you use for your application.
- You can use and link information from different databases or applications. For each application you can make a different DID module.

Using the import/export mechanism of the KCM DID specification language, you can link different DID modules together.

You cannot declare an imported entry to be a main entry.

Syntax

```
alias-definition ::=
ALIAS_ENTRY entry-name ALIAS_OF entry-name
import-definition ::=
IMPORT_ENTRY entry-name IMPORT_FROM did-module-name
```

Example

```
ALIAS_ENTRY Postal_address_of_customer ALIAS_OF Address
IMPORT_ENTRY Customer IMPORT_FROM Relations_module
```

Fields

A Field is a means to contain access data retrieved by an entry.

Syntax

```
field-sequence ::=
field-def [ field-sequence ]
field-def ::=
field-name field-type-description
field-type-description ::=
file-field-type
[ database-field-name ]
[ screen-field-def ]
[ as400 connection field-attributes ]
file-field-type ::=
ZONED field-type-length-double
| PACKED field-type-length-double
| BINARY field-type-length-double
| NUMERICAL field-type-length-double
| DOUBLE
| TEXT field-type-length-single [ alignment_info ]
| C_CHAR field-type-length-single
| W_CHAR field-type-length-single
field-type-length-double ::=
[ LENGTH ] '(' natural-number  natural-number  ')'
field-type-length-single ::=
[ LENGTH ] '(' natural-number  ')'
alignment-info ::=
[ ALIGNMENT ] direction-alignment
direction-alignment ::=
LEFT_ADJUSTED
| RIGHT_ADJUSTED
database-field-name ::=
DATABASE_FIELD field-specification
screen-field-def ::=
SCREEN_FIELD natural-number
as400-connection-field-attributes ::=
database-offset
database-offset ::=
OFFSET natural-number
```

Example

```
Customer_number NUMERICAL LENGTH(10 0) DATABASE_FIELD "CUSTNO"
Customer_surname TEXT LENGTH(50)   DATABASE_FIELD "CUSTSNAME"
Customer_name TEXT LENGTH(50)   DATABASE_FIELD "CUSTNAME"
```

Field attributes

All Fields have the following attributes:

- Name

  Sets the name of the Field. This name is used in the Master Template development. The name must begin with an uppercase character and can be followed by lowercase characters, uppercase characters, digits, and underscores.

  ```
  field-name ::=
  uppercase-itp-name
  ```

- Type

  The type consists of a type name, an optional length definition, and optional alignment information. The KCM DID specification language allows you to use all types of every connection in DID modules. These types are the Field types that KCM retrieves and converts during the Master Template execution by mapping the values to the type values of the Template scripting language: `NUMBER`, `TEXT`, and `BOOL`.

- Length

  Length definition depends on the Field type. Some numerical types have a double length definition. In this case, the first parameter indicates the byte size of the Field, and the second parameter indicates the number of decimals. For null-terminated data types, such as `C_CHAR`, `W_CHAR`, or `NUMERICAL`, the null character should be included in the byte-count.

- Alignment

  Fields can contain alignment information. When a Field is left-aligned, KCM strips spaces at the end of the Field. When a Field is right-aligned, KCM strips spaces at the beginning of the Field. Left-alignment is used by default.

- Database Field

  Stores names of Fields and columns of database management systems. Some database management systems use naming conventions that yield names that are invalid according to the KCM naming convention. In some connection types, KCM queries the database management system directly. Therefore, KCM needs the information on how the Field or column is known in the database management system. The value of the database Field attribute can be any string that is a valid Field or column specification. The connection type determines if values are valid.

- Screen Field

  Some connection types use this attribute to construct a key retrieval definition automatically. The screen Field attribute expects a natural number that specifies the order in which the screen Fields are presented. The lower numbers are parsed first. When you specify the same number twice, the order for the Fields is determined by KCM.

In addition to this set of parameters, a Field can have a calling connection attribute that depends on the connection type specified.

Chapter 4

Configure connection types

This chapter lists and describes the connection types that you can configure in a DID module of a KCM DID. The following types are currently supported:

- Local connection for local programs and DLLs
- ODBC connection for ODBC enabled databases
- Oracle connection for Oracle databases with SQL*Net
- XML file connection for retrieving data from XML files
- XML Web Services connection for retrieving data from XML files
- XML MQSeries connection for retrieving data from XML files

Specify a connection type for a DID

Use KCM Core Administrator to specify a connection type for a KCM DID. You can set three connection configuration levels:

1. For all DIDs containing entries, using a specific connection type
2. For each DID
3. For each DID module

Some settings also have a fourth level: you can specify command line parameters when using the APIs.

KCM searches these levels in reverse order, starting with the command line parameters, and uses the first value it encounters. If it cannot find a value for a specific setting, it uses a default. Otherwise, an error is generated.

You can combine several connection types in a single DID. Each DID module uses its own connection type. Consequently, you can combine settings for multiple connection types in the DID settings and have more than one connection to a database, data source or server per KCM connection.

Local connection

With the local connection, you can retrieve data through calls to programs or functions in dynamic link libraries (DLL) that run on the server.

When you run a Master Template, the Data Manager calls programs and functions in the respective DLL files. Specify the location of these files in the connection configuration in KCM Core Administrator.

Usage

To use this connection type, follow these steps:

1. Open the required DID document and locate a DID module that contains local connection entries.
   A DID can contain multiple local connection modules.
2. Set the `CONNECTION` attribute to `LOCAL`.
3. Set the `MODEL_DOC_STATEMENT` to `WITH` for a single entry to retrieve only one record or `FORALL` for an entry where zero, one, or more records need to be retrieved.
4. Set the `CALLING_CONVENTION` attribute of an entry to one of the following values:
   - For a program, set to `PROGRAM`. Specify the name of the program to call in the `DATA_RETRIEVAL` or `KEY_RETRIEVAL` attributes.
   - For a 32-bit or 16-bit DLL, set to `DLL32` or `DLL16`, respectively. Specify the name of the function to call in the `DATA_RETRIEVAL` or `KEY_RETRIEVAL` attributes. The name must follow this form: `<functionname>@<dllname>`. For example, `Customer@Example.dll`.

Example

```
DEFINE_ENTRY
   NAME                    MyEntry
   MODEL_DOC_STATEMENT     WITH
   CALLING_CONVENTION      PROGRAM
   DATA_RETRIEVAL          "MyProg.exe"
   KEY_RETRIEVAL           "*NONE"
```

In this example, a local connection entry `MyEntry` uses a program `MyProg.exe` to retrieve data.

Example

```
DEFINE_ENTRY
   NAME                    MyEntry
   MODEL_DOC_STATEMENT     WITH
   CALLING_CONVENTIONS     DLL32
   DATA_RETRIEVAL          "MyEntry@MyDll.dll"
   KEY_RETRIEVAL           "*NONE"
```

In this example, a local connection entry `MyEntry` uses a function in a 32-bit DLL to retrieve data.

All data types are supported for Fields in a local connection entry.

Sample local connection entry

The following example illustrates a local connection entry named `Customer`.

The entry is used in a Master Template.

```
WITH Customer IN DID.Customer
DO
…
OD
```

The entry as defined in the DID document. The entry definition assumes that the C code is implemented in a DLL `Example.dll`.

```
DEFINE_ENTRY
```

```
   NAME                Customer
   MODEL_DOC_STATEMENT WITH
   DATA_RETRIEVAL      "Customer@Example.dll"
   KEY_RETRIEVAL       "*NONE"
   CALLING_CONVENTION  DLL32
   DEFINE_PARAMETERS
      Customer_number     NUMERICAL ( 9 0)
   END_DEFINE_PARAMETERS
   DEFINE_FIELDS
      Customer_number     NUMERICAL ( 9 0)
      Surname             C_CHAR    (31  )
      First_name          C_CHAR    (31  )
      Initials            C_CHAR    (11  )
      Date_of_birth       NUMERICAL (11 0)
      Date_of_death       NUMERICAL (11 0)
      Gender              C_CHAR    ( 2  )
      Marital_state       C_CHAR    ( 2  )
      Partner_number      NUMERICAL ( 9 0)
   END_DEFINE_FIELDS
END_DEFINE_ENTRY   (* Customer *)
```

The following is a C code sample for data retrieval for the entry `Customer`. This code is generated by the C code wizard. At the comment `Fill variable(s)`, insert your own code to retrieve the contents of the result fields.

```
/*
 * Function: Customer
 */
#include "itpentry.h"
#include "itpcnv.h"
#ifdef __cplusplus
extern "C" {
  int __stdcall _export Customer(void);
}
#endif
int __stdcall _export Customer(void){
   int rc;
   /* Input definition */
   unsigned char InputRecord[1+1+9];
   int           Customernumber;
   /* Output definition */
   unsigned char OutputRecord[117];
   int           Customernumber;
   unsigned char Surname[31 + 1];
   unsigned char Firstname[31 + 1];
   unsigned char Initials[11 + 1];
   int           Dateofbirth;
   int           Dateofdeath;
   unsigned char Gender[2 + 1];
   unsigned char Maritalstate[2 + 1];
   int           Partnernumber;
   /* Initialize entryapi */
   APICALL(ITPAPIR(3));
   /* Read input parameter block */
   APICALL(ITPRCV(InputRecord, sizeof(InputRecord)));
   Customernumber = GetNumericalField(InputRecord+2, 9);
   /* Prepare for sending output to ITP */
   APICALL(ITPOPEN());
   /* Prepare and send output (record(s)) to ITP */
   ..... /* Fill variable(s) */
   PutNumericalField(OutputRecord +    0, Customernumber);
   PutTextField(OutputRecord +    9, Surname, 31);
```

```
   PutTextField(OutputRecord +   40, Firstname, 31);
   PutTextField(OutputRecord +   71, Initials, 11);
   PutNumericalField(OutputRecord +   82, Dateofbirth);
   PutNumericalField(OutputRecord +   93, Dateofdeath);
   PutTextField(OutputRecord +  104, Gender, 2);
   PutTextField(OutputRecord +  106, Maritalstate, 2);
   PutNumericalField(OutputRecord +  108, Partnernumber);
   APICALL(ITPSND(OutputRecord, sizeof(OutputRecord)));
   /* Close and return to ITP */
   APICALL(ITPCLOSE(0));
   return 0;
}
```

ODBC connection

With the ODBC connection, you can retrieve data from an Open Database Connectivity data source using SQL statements. The SQL statements, input parameters, and output columns are all part of the entry definition. The SQL statements and functions to use depend on the data source type and the ODBC driver.

When you run a Master Template, the Data Manager connects to an ODBC data source. Specify the data source to connect to in the connection configuration in KCM Core Administrator.

To connect to multiple data sources in a single DID, you can specify the ODBC settings per DID module. If you omit settings for a certain DID module, KCM uses the settings specified at the DID level for this DID module.

For information about ODBC connection strings, see the documentation for your ODBC driver.

Requirements

The ODBC driver must be an ODBC 2.0 driver or higher, and it must implement all core functions and the following level one functions:

- SQLBindParameter
- SQLDriverConnect

The KCM DID Development Kit comes with a wizard to generate an ODBC entry. This wizard requires additional level one and level two functions.

Also, consider the following limitations when using this connection type:

- You cannot use stored functions.
- Use the SQL `SELECT` and `CALL` statements in the data and key retrieval statements.
- To update the database, use the `UPDATE` statement to create DID defined functions.

Usage

To use this connection type, follow these steps:

1. Open the required DID document and locate a DID module that contains an ODBC connection entry.

   A DID can contain multiple ODBC connection modules. However, KCM can connect to only one data source at a given time. You can use different data sources for different DIDs and Master Templates, but not within a single DID.

2. Set the `CONNECTION` attribute to `ODBC`.
3. To construct the `SELECT` part of an SQL statement, specify a column name in the `DATABASE_FIELD` attribute. You can rename a database column name to a KCM Field name.

   In this example, a database column named `ORDNR` is renamed to a Field name `Order_number`.

   ```
   Order_number NUMERICAL(8 0) DATABASE_FIELD "ORDNR"
   ```

   The following data types are supported for Fields in the ODBC connection: `C_CHAR`, `TEXT`, `NUMERICAL`, and `DOUBLEW_CHA`.

Sample ODBC connection entry

This example illustrates an ODBC connection entry named `Customer`.

```
(*
 * Entry:              Customer
 * SQL Table:          CUSTOMER
 *)
DEFINE_ENTRY
   NAME                Customer
   MODEL_DOC_STATEMENT WITH
   DATA_RETRIEVAL      "from CUSTOMER where CUSTOMERNUMBER=:1"
   KEY_RETRIEVAL       "from CUSTOMER where (CUSTOMERNUMBER>=:1)"
   DEFINE_PARAMETERS
      Customernumber   NUMERICAL ( 9 0) DATABASE_FIELD "CUSTOMERNUMBER"
   END_DEFINE_PARAMETERS
   DEFINE_FIELDS
      Customernumber   NUMERICAL ( 9 0) DATABASE_FIELD "CUSTOMERNUMBER"
      Surname          C_CHAR    (31  ) DATABASE_FIELD "SURNAME" SCREEN_FIELD 1
      Firstname        C_CHAR    (31  ) DATABASE_FIELD "FIRSTNAME"
      Initials         C_CHAR    (11  ) DATABASE_FIELD "INITIALS"
      Dateofbirth      NUMERICAL (11 0) DATABASE_FIELD "( {fn year(DATEOFBIRTH)} *
 10000 + {fn month(DATEOFBIRTH)} * 100 + {fn dayofmonth(DATEOFBIRTH)} )"
      Dateofdeath      NUMERICAL (11 0) DATABASE_FIELD "( {fn year(DATEOFDEATH)} *
 10000 + {fn month(DATEOFDEATH)} * 100 + {fn dayofmonth(DATEOFDEATH)} )"
      Gender           C_CHAR    ( 2  ) DATABASE_FIELD "GENDER"
      Maritalstate     C_CHAR    ( 2  ) DATABASE_FIELD "MARITALSTATE"
      Partnernumber    NUMERICAL ( 9 0) DATABASE_FIELD "PARTNERNUMBER"
   END_DEFINE_FIELDS
   DEFINE_SUBENTRIES
      (* Place subentry definitions here *)
   END_DEFINE_SUBENTRIES
END_DEFINE_ENTRY   (* Customer *)
```

Oracle connection

With the Oracle connection, you can retrieve data from an Oracle database using SQL statements. The SQL statements, input parameters, and output columns are all part of the entry definition.

When you run a Master Template, the Data Manager connects to an Oracle database. Specify the database to connect to in the connection configuration in KCM Core Administrator.

Requirements

To use the Unicode support, KCM requires Oracle Client 8.0 or higher.

Also, consider the following limitations when using this connection type:

- Use the SQL `SELECT` statement in the data and key retrieval statements.
- You cannot use stored procedures or stored functions.
- To create DID defined functions, use the `UPDATE` statement.
- To update the database, use formal parameters of type `W_CHAR` if the corresponding columns are specified as Unicode in the database.

Usage

To use this connection type, follow these steps:

1. Open the required DID document and locate a DID module that contains Oracle connection entries.

   A DID can contain multiple Oracle connection modules. However, KCM can connect to only one Oracle database at a time. You can use different databases for different DIDs and Master Templates, but not within a single DID.

2. Set the `CONNECTION` attribute to `ORACLE`.

3. To construct the `SELECT` part of an SQL statement, specify a column name in the `DATABASE_FIELD` attribute. You can rename a database column name to a KCM Field name.

   In this example, a database column named `ORDNR` is renamed to a Field name `Order_number`.

   ```
   Order_number NUMERICAL(8 0) DATABASE_FIELD "ORDNR"
   ```

The following data types are supported for Fields in the Oracle connection: `C_CHAR`, `NUMERICAL`, and `W_CHAR`.

Sample Oracle connection entry

This example illustrates an Oracle connection entry named `Customer`.

```
(*
 * Entry:                Customer
 * SQL Table:            CUSTOMER
 *
 *)
DEFINE_ENTRY
   NAME                Customer
   MODEL_DOC_STATEMENT WITH
   DATA_RETRIEVAL      "from CUSTOMER where CUSTOMERNUMBER=:1"
   KEY_RETRIEVAL       "from CUSTOMER where (CUSTOMERNUMBER>=:1)"
   DEFINE_PARAMETERS
      Customernumber   NUMERICAL ( 9 0) DATABASE_FIELD "CUSTOMERNUMBER"
   END_DEFINE_PARAMETERS
   DEFINE_FIELDS
      Customernumber   NUMERICAL ( 9 0) DATABASE_FIELD "CUSTOMERNUMBER"
      Surname          C_CHAR    (31  ) DATABASE_FIELD "SURNAME" SCREEN_FIELD 1
      Firstname        C_CHAR    (31  ) DATABASE_FIELD "FIRSTNAME"
      Initials         C_CHAR    (11  ) DATABASE_FIELD "INITIALS"
      Dateofbirth      NUMERICAL (11 0) DATABASE_FIELD "(to_char(DATEOFBIRTH,
 'YYYYMMDD'))"
      Dateofdeath      NUMERICAL (11 0) DATABASE_FIELD "(to_char(DATEOFDEATH,
 'YYYYMMDD'))"
      Gender           C_CHAR    ( 2  ) DATABASE_FIELD "GENDER"
      Maritalstate     C_CHAR    ( 2  ) DATABASE_FIELD "MARITALSTATE"
      Partnernumber    NUMERICAL ( 9 0) DATABASE_FIELD "PARTNERNUMBER"
   END_DEFINE_FIELDS
```

```
   DEFINE_SUBENTRIES
      (* Place subentry definitions here *)
   END_DEFINE_SUBENTRIES
END_DEFINE_ENTRY   (* Customer *)
```

XML file connection

With the XML file connection, you can retrieve data from an XML file using XML tags. The tags and output columns are part of the entry definition.

You can pass the XML file as the `KEYS` argument to the RUNMDL API. The other settings are specified in the connection configuration in KCM Core Administrator.

By default, XML files use UTF-8 encoding.

Usage

To use this connection type, follow these steps:

1. Open the required DID document and locate a DID module that contains an XML file connection entry.

   A DID can contain multiple XML file connection modules. You can specify different default folders where KCM should start looking for XML files. To do so, use the `Directory` setting in the connection configuration in KCM Core Administrator.
2. Set the `CONNECTION` attribute to `XMLFILE`.
3. Set the `DATA_RETRIEVAL` attribute of an entry to the XML tag that identifies the entry in the XML file.
4. Set the `MODEL_DOC_STATEMENT` attribute of the entry to `WITH` for a single entry to retrieve the first record with a matching tag or `FORALL` to retrieve all records with a matching tag.

The following data types are currently supported for Fields in this connection type: `C_CHAR`, `W_CHAR`, and `DOUBLE`.

Sample DID document with XML file connection

```
DEFINE_DID
  NAME            "XMLexampleDID"
  IDENTIFICATION  XML
  DIDMODULE_LIST  "XMLexampleDID"
  MAIN_ENTRY_LIST
        Customer                        FROM "XMLexampleDID"
END_DEFINE_DID
DEFINE_DIDMODULE
  NAME       "XMLexampleDID"
  CONNECTION XMLFILE
DEFINE_ENTRY
    NAME                      Customer                        (* Customer *)
    MODEL_DOCUMENT_STATEMENT  WITH
    DATA_RETRIEVAL            "Customer"
    KEY_RETRIEVAL             "*NONE"
    DEFINE_FIELDS
```

```
      CustomerNumber                  DOUBLE                    DATABASE_FIELD
 "CustomerNumber"
      LastName                        C_CHAR  LENGTH ( 255  ) DATABASE_FIELD "LastName"
      Prefix                          C_CHAR  LENGTH ( 255  ) DATABASE_FIELD "@Prefix"
      FirstName                       C_CHAR  LENGTH ( 255  ) DATABASE_FIELD "FirstName"
      Initials                        C_CHAR  LENGTH ( 255  ) DATABASE_FIELD "Initials"
      DateOfBirth                     DOUBLE                    DATABASE_FIELD
 "DateOfBirth"
      DateOfDeath                     DOUBLE                    DATABASE_FIELD
 "DateOfDeath"
      Gender                             C_CHAR  LENGTH ( 255  ) DATABASE_FIELD "Gender"
      MaritialStatus                  C_CHAR  LENGTH ( 255  ) DATABASE_FIELD
 "MaritialStatus"
      PartnerNumber                   DOUBLE                    DATABASE_FIELD
 "PartnerNumber"
      Info                            C_CHAR  LENGTH ( 255  ) DATABASE_FIELD "Info"
       Comment                         C_CHAR  LENGTH ( 255  ) DATABASE_FIELD "Comment"
(* Info and Comment are mixed content elements, they are fields of their containing tag
 as well as subentries. See below for their subentry definition *)
     END_DEFINE_FIELDS
    DEFINE_SUBENTRIES
      Address                         (* Address *)
      Info                            (* Mixed content element; both a field of the
                                         containing tag and a subentry *)
      Comment                         (* Mixed content element; both a field of the
                                         containing tag and a subentry *)
    END_DEFINE_SUBENTRIES
  END_DEFINE_ENTRY
  DEFINE_ENTRY
    NAME                      Address                         (* Address *)
    MODEL_DOCUMENT_STATEMENT FORALL
    DATA_RETRIEVAL           "Address"
    KEY_RETRIEVAL            "*NONE"
    DEFINE_FIELDS
      CustomerNumber                  DOUBLE                    DATABASE_FIELD
 "CustomerNumber"
      ZipCode                         DOUBLE                    DATABASE_FIELD "ZipCode"
      Number                          DOUBLE                    DATABASE_FIELD "Number"
      Street                          C_CHAR  LENGTH ( 255  ) DATABASE_FIELD "Street"
      City                            C_CHAR  LENGTH ( 255  ) DATABASE_FIELD "City"
      Country                         C_CHAR  LENGTH ( 255  ) DATABASE_FIELD "Country"
    END_DEFINE_FIELDS
    DEFINE_SUBENTRIES
    END_DEFINE_SUBENTRIES
  END_DEFINE_ENTRY
DEFINE_ENTRY
    NAME                      Info                            (* This is the subentry
 Info *)
    MODEL_DOCUMENT_STATEMENT WITH
    DATA_RETRIEVAL           "Info"
    KEY_RETRIEVAL            "*NONE"
    DEFINE_FIELDS
      Source                        C_CHAR LENGTH ( 255  )  DATABASE_FIELD "Source"
      Source_date                   C_CHAR LENGTH ( 255  )  DATABASE_FIELD "SourceDate"
 (* These elemnts are the non-repeating tags within Info, turned into fields of entry
 Info *)
    END_DEFINE_FIELDS
    DEFINE_SUBENTRIES
    END_DEFINE_SUBENTRIES
  END_DEFINE_ENTRY
DEFINE_ENTRY
    NAME                      Comment                            (* Comment *)
    MODEL_DOCUMENT_STATEMENT WITH
    DATA_RETRIEVAL           "Comment"
```

```
    KEY_RETRIEVAL            "*NONE"
    DEFINE_FIELDS
    END_DEFINE_FIELDS
    DEFINE_SUBENTRIES
      P                               (* p *)
      (* This is the repeating tag within the mixed content 'Comment' tag and a
         subentry of Comment.*)
    END_DEFINE_SUBENTRIES
  END_DEFINE_ENTRY
DEFINE_ENTRY
NAME                     P         (* definition of the subentry P containing the
                                      repeating tag within Comment *)
    MODEL_DOCUMENT_STATEMENT FORALL
    DATA_RETRIEVAL           "p"
    KEY_RETRIEVAL            "*NONE"
    DEFINE_FIELDS
      Self                           C_CHAR LENGTH ( 255  )  DATABASE_FIELD "."
(* This field provides access to the values contained in the repeating subtag 'p' *)
    END_DEFINE_FIELDS
    DEFINE_SUBENTRIES
    END_DEFINE_SUBENTRIES
  END_DEFINE_ENTRY
END_DEFINE_DIDMODULE
```

XML Web Services connection

With the XML Web Services connection, you can retrieve data from XML through the HTTP protocol. As different HTTP servers expect different types of messages, the XML Web Services connection is set up in a flexible way. You can control the overall layout of the HTTP request to a large extent with templates. This applies both to HTTP header and to the content of the message. This allows for the construction of any type of HTTP request message, including messages without XML data.

When an entry is executed in the Master Template, the XML Web Services connection either extracts data from an earlier collected XML structure sends an HTTP request through an URL. The `HostName`, `Port`, and `URI` from which this URL is composed can be configured in the connection configuration in KCM Core Administrator.

You can specify a template for the content of the HTTP message and a template for its headers. If a content template is specified, the content type of the HTTP message needs to be specified as well (see Example content template).

Also, you can specify an error tag in the connection configuration to include error information.

By default, XML files use UTF-8 encoding.

Usage

To use this connection type, follow these steps:

1. Open the required DID document and locate a DID module that contains XML Web Services connection entries.
2. Set the `CONNECTION` attribute to `XMLWEB`.

3. Set the CALLING_CONVENTION attribute to one of the following values:
   - HTTPREQUEST. These entries collect data through an actual HTTP request.
   - SUBELEMENT. These entries collect data through XML structures obtained from the Master Template.
4. The DATA_RETRIEVAL attribute is dependent on the value set in the CALLING_CONVENTION attribute:
   - If CALLING_CONVENTION is set to HTTPREQUEST, the DATA_RETRIEVAL holds the function that is being called. The returned records are specified at the DEFINE_FIELDS section of the entry. You can refer to the DATA_RETRIEVAL key from within the content template with the expression @(ENTRY:RETRIEVAL).
   - If CALLING_CONVENTION is set to SUBELEMENT, the DATA_RETRIEVAL key specifies the XML tag that identifies the entry in the XML file. The returned records are specified at the DEFINE_FIELDS section of the entry. XML tags are case-sensitive.

   KEY_RETRIEVAL is only possible for entries that collect their data through MQSeries. It holds the function that is being called. The returned records are specified in a key at the DEFINE_PARAMETERS section of the entry that describes the parameters of the function.

   You can refer to the KEY_RETRIEVAL key from within the content template with the expression @(ENTRY:RETRIEVAL).
5. Set the MODEL_DOC_STATEMENT to WITH for a single entry to retrieve only one record or FORALL for an entry where zero, one, or more records need to be retrieved.
6. Set the DEFINE section to keys to refer to in the content template. These keys describe the parameters for the key and data retrieval. This gives you the ability to pass any value from the DID to the template.

The result of the HTTP request is returned in an XML structure. The RESULT_PATH specifies where the KEY_RETRIEVAL and DATA_RETRIEVAL are located in this XML structure. Each item in the path specifies the tag of an XML element. The first one holds the root element and the last one specifies the tag of the returned records.

```
RESULT_PATH
"biztalk_1//body//@(ENTRY:RETRIEVAL).Response//EMPLOYEE//item"
```

You can include references to the function name with the @(ENTRY:RETRIEVAL) statements. Slashes must be escaped. You can use backslashes instead as shown in this example.

```
RESULT_PATH
"biztalk_1\body\@(ENTRY:RETRIEVAL).Response\Employee\item"
```

Also, you can include @(PAR:…) and general @(ENTRY:…) statements when the actual function name is passed as a parameter (and part of the result path).

The result path is not parsed for namespace prefixes. Any prefixes must be included in the path.

Retrieving attributes

An attribute of an XML element becomes a field of an entry with the same name as the element tag.

Example

```
<Root>
      <Var1 Attribute="x">
            Content of var1.
      </Var1>
 </Root>
```

This XML is specified in the entry in the following way:

```
DEFINE_ENTRY
    NAME                      Root                            (* Root *)
    MODEL_DOCUMENT_STATEMENT WITH
      DEFINE_FIELDS
        Var1                C_CHAR LENGTH ( 255  )  DATABASE_FIELD "Var1"
      END_DEFINE_FIELDS
      DEFINE_SUBENTRIES
        Var1                    (* Var1 *)
      END_DEFINE_SUBENTRIES
  END_DEFINE_ENTRY
  DEFINE_ENTRY
    NAME                      Var1                            (* Var1 *)
    MODEL_DOCUMENT_STATEMENT WITH
      DEFINE_FIELDS
        Attribute           C_CHAR LENGTH ( 255  )  DATABASE_FIELD "@Attribute"
      END_DEFINE_FIELDS
  END_DEFINE_ENTRY
```

Set the `DATABASE_FIELD` for such attribute to `@<name of the attribute>` to retrieve the content of the attribute.

Tip You can retrieve the content of a current node with a single period.

The following data types are supported for Fields in the XML Web Services connection: `C_CHAR`, `W_CHAR`, and `DOUBLE`.

Sample DID document with XML Web Services connection

In the following example DID document, the `DATA_RETRIEVAL` and `KEY_RETRIEVAL` keys hold the name of the function called by the HTTP server. The `DEFINE` section holds a `PARAMS` definition that maps the parameters of the entry to the format needed for the specific function. The `RESULT_PATH` determines the path in the XML tree to the result of the function.

You can refer to the keys in the content template with the `@(ENTRY:<name_key>)` construction. The keys that the Example content template refers to are made bold in this DID document.

```
DEFINE_DID
   NAME           "SAPEXAMPLE"
   IDENTIFICATION SAP
   DIDMODULE_LIST
      "SAPMODULE"
   MAIN_ENTRY_LIST
 Customer FROM  SAPMODULE
END_DEFINE_DID (* "SAPEXAMPLE" *)
DEFINE_DIDMODULE
   NAME           "SAPMODULE"
   CONNECTION     XMLWEB
 DEFINE_ENTRY
  NAME Customer
  MODEL_DOC_STATEMENT WITH
  DATA_RETRIEVAL "GetCustomer"
```

```
  KEY_RETRIEVAL "GetCustomerKeys"
  CALLING_CONVENTION HTTPREQUEST
  RESULT_PATH "biztalk_1//body//@(ENTRY:RETRIEVAL)//CUSTOMER//item"
 DEFINE
  PARAMS "<CUST_ID>@(PAR:CustomerNumber)</CUST_ID>"
  COMMENT "Get the customer with the passed parameter"
 END_DEFINE
 DEFINE_PARAMETERS
  CustomerNumber C_CHAR LENGTH(255)
 END_DEFINE_PARAMETERS
 DEFINE_FIELDS
  Customer_number C_CHAR LENGTH(255) DATABASE_FIELD CUST_ID
  Prefix   C_CHAR LENGTH(255) DATABASE_FIELD @PREFIX
  Surname  C_CHAR LENGTH(255) DATABASE_FIELD SURNAME
  Initials  C_CHAR LENGTH(255) DATABASE_FIELD INITIALS
  Gender   C_CHAR LENGTH(255) DATABASE_FIELD GENDER
 END_DEFINE_FIELDS
 END_DEFINE_ENTRY
END_DEFINE_DIDMODULE (* "SAPMODULE" *)
```

XML MQSeries connection

With the XML MQSeries connection, you can retrieve data from an XML file through MQSeries. As different servers expect different types of messages, the XML MQSeries connection is set up in a flexible way. You can control the overall layout of the MQSeries request to a large extent with a template. This allows for the construction of any type of MQSeries request message, including messages without XML data.

When an entry is executed in the Master Template, the XML MQSeries connection either extracts data from an earlier collected XML structure, or sends an MQSeries request.

You can configure the XML MQSeries connection in the connection configuration in KCM Core Administrator. Also, you can specify an error tag in the connection configuration to include error information.

By default, XML files use UTF-8 encoding.

Usage

To use this connection type, follow these steps:

1. Open the required DID document and locate a DID module that contains an XML Web Services connection entry.
2. Set the `CONNECTION` attribute to `MQXML`.
3. Set the `CALLING_CONVENTION` attribute to one of the following values:
   - `MQREQUEST`. These entries collect data through an actual MQSeries request.
   - `SUBELEMENT`. These entries collect data through XML structures obtained from the Master Template.
4. The `DATA_RETRIEVAL` attribute is dependent on the value set in the `CALLING_CONVENTION` attribute:
   - If `CALLING_CONVENTION` is set to `MQREQUEST`, the `DATA_RETRIEVAL` holds the function that is being called. The returned records are specified at the `DEFINE_FIELDS` section of the entry.

You can refer to the DATA_RETRIEVAL key from within the content template with the expression @(ENTRY:RETRIEVAL).

- If CALLING_CONVENTION is set to SUBELEMENT. The DATA_RETRIEVAL key specifies the XML tag that identifies the entry in the XML file. The returned records are specified at the DEFINE_FIELDS section of the entry. XML tags are case-sensitive.

KEY_RETRIEVAL is only possible for entries that collect their data through MQSeries. It holds the function that is being called. The returned records are specified in a key at the DEFINE_PARAMETERS section of the entry that describes the parameters of the function.

You can refer to the KEY_RETRIEVAL key from within the content template with the expression @(ENTRY:RETRIEVAL).

5. Set the MODEL_DOC_STATEMENT to WITH for a single entry to retrieve only one record or FORALL for an entry where zero, one, or more records need to be retrieved.
6. Set the DEFINE section to keys to refer to in the content template. These keys describe the parameters for the key and data retrieval. This gives you the ability to pass any value from the DID to the template.

The result of the HTTP request is returned in an XML structure. The RESULT_PATH specifies where the KEY_RETRIEVAL and DATA_RETRIEVAL are located in this XML structure. Each item in the path specifies the tag of an XML element. The first one holds the root element and the last one specifies the tag of the returned records.

```
RESULT_PATH "biztalk_1//body//@(ENTRY:RETRIEVAL).Response//EMPLOYEE//item"
```

You include references to the function name with the @(ENTRY:RETRIEVAL) statements. Slashes must be escaped. You can use backslashes instead, as shown in this example.

```
RESULT_PATH "biztalk_1\body\@(ENTRY:RETRIEVAL).Response\Employee\item"
```

Also, you can include @(PAR:…) and general @(ENTRY:…) statements when the actual function name is passed as a parameter (and part of the result path).

The result path is not parsed for namespace prefixes. Any prefixes must be included in the path.

Retrieving attributes

An attribute of an XML element becomes a field of an entry with the same name as the element tag.

```
<Root>
     <Var1 Attribute="x">
          Content of var1.
     </Var1>
</Root>
```

This XML is specified in the entry in the following way:

```
DEFINE_ENTRY
    NAME                      Root                          (* Root *)
    MODEL_DOCUMENT_STATEMENT WITH
      DEFINE_FIELDS
        Var1                C_CHAR LENGTH ( 255  )  DATABASE_FIELD "Var1"
      END_DEFINE_FIELDS
```

```
      DEFINE_SUBENTRIES
        Var1                    (* Var1 *)
      END_DEFINE_SUBENTRIES
  END_DEFINE_ENTRY
  DEFINE_ENTRY
    NAME                      Var1                            (* Var1 *)
    MODEL_DOCUMENT_STATEMENT WITH
      DEFINE_FIELDS
        Attribute           C_CHAR LENGTH ( 255  )  DATABASE_FIELD "@Attribute"
      END_DEFINE_FIELDS
  END_DEFINE_ENTRY
```

The `DATABASE_FIELD` for such attribute needs to be set to `@<name of the attribute>` to retrieve the content of the attribute.

Tip You can retrieve the content of a current node with a single period.

Only the following data types are supported for Fields in the XML MQSeries connection: `C_CHAR`, `W_CHAR`, and `DOUBLE`.

Sample DID document with the XML MQSeries connection

In the following example DID document, the `DATA_RETRIEVAL` and `KEY_RETRIEVAL` keys can be used to distinguish the type of request (data retrieval or key retrieval). The `DEFINE` section holds a `PARAMS` definition that maps the parameters of the entry to the format needed for the specific function. The `RESULT_PATH` determines the path in the XML tree to the result of the function.

You can refer to the keys in the content template with the `@(ENTRY:<name_key>)` construction. The keys that the example content template refers to are bold in the following sample DID document.

```
DEFINE_DID
   NAME           "SAPEXAMPLE"
   IDENTIFICATION SAP
   DIDMODULE_LIST
      "SAPMODULE"
   MAIN_ENTRY_LIST
 Customer FROM  SAPMODULE
END_DEFINE_DID (* "SAPEXAMPLE" *)
DEFINE_DIDMODULE
   NAME           "SAPMODULE"
   CONNECTION     MQXML
 DEFINE_ENTRY
  NAME Customer
  MODEL_DOC_STATEMENT WITH
  DATA_RETRIEVAL "GetCustomer"
  KEY_RETRIEVAL "GetCustomerKeys"
  CALLING_CONVENTION MQREQUEST
  RESULT_PATH "biztalk_1//body//@(ENTRY:RETRIEVAL)//CUSTOMER//item"
 DEFINE
  PARAMS "<CUST_ID>@(PAR:CustomerNumber)</CUST_ID>"
  COMMENT "Get the customer with the passed parameter"
 END_DEFINE
 DEFINE_PARAMETERS
  CustomerNumber  C_CHAR LENGTH(255)
 END_DEFINE_PARAMETERS
 DEFINE_FIELDS
  Customer_number C_CHAR LENGTH(255)

DATABASE_FIELD ("CustomerNumber")
```

```
  Prefix   C_CHAR LENGTH(255)
  Surname   C_CHAR LENGTH(255)
  Initials  C_CHAR LENGTH(255)
  Gender   C_CHAR LENGTH(255)
 END_DEFINE_FIELDS
 END_DEFINE_ENTRY
END_DEFINE_DIDMODULE (* "SAPMODULE" *)
```

Example content template

In the content template, you specify the format for the content of the XML message. References to the entry keys are made bold in this example.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<biztalk_1 xmlns="urn:biztalk-org:biztalk:biztalk_1">
 <header>
  <delivery>
   <to>
    <address>@(DM:To)</address>
   </to>
   <from>
    <address>@(DM:From)</address>
   </from>
  </delivery>
 </header>
 <body>
  <!--@(ENTRY:COMMENT)-->
  <@(ENTRY:RETRIEVAL) xmlns:doc="urn:sap-com:document:sap:rfc:functions" mlns="">
   @(ENTRY:PARAMS)
  </@(ENTRY:RETRIEVAL)>
 </body>
</biztalk_1>
```