5. DCS Business Object Ontology

The DCS operates on a set of object models called Business Objects. This section enumerates the types of Business Objects and describes the relationships supported amongst them. In addition to the specified relationships, relationships that do not violate the described cardinalities/constraints may be captured through a generic relationship mechanism.

For data packaging purposes, each object type can be represented as a ORE Resource Map under the resource map profile explained here. The tables for each object type indicate the mapping between the properties and relationships of the object models and their corresponding ORE ReM representations. Some properties and relationships are not included in the ReMs; therefore, they have no ORE-ReM representation.

5.1. Project Model

Project

Usage

Generally speaking, a Project is meant to align with a topic or area of sponsored research (e.g. a grant, or faculty research project). It records information important to data managers, such as the sponsors of a project, the administrators (e.g. the Principle Investigators and/or their proxies), and the beginning and end dates of the research project.

Model

Property	Cardinality	ORE-ReM representation	Usage
Id	(1,1)	dcterms:identifier	Generated by the system during ingest (via package ingest) or creation (via the DCS Front End)
Name	(1,1)	dcterms:title	Brief textual title for the Project
Description	(1,1)	dcterms:description	Textual description of the Project
Funding Entity	(0,1)		Identifies funder
Numbers	(0,*)		Award or other funding "numbers".
Principal Investigators	(0,*)		Identities (names) of principal investigators
Publisher	(1,1)		
Start Date	(1,1)		The start date of the Project
End Date	(0,1)		The end date of the Project
Storage Allocated	(1,1)		Storage space allotted to the Project in the system

Storage Used	(1,1)		Storage space taken up by the Project's data.

Relationship name	Cardinality	Targets	ORE-ReM representation	Usage
hasMember	(0,*)	Collection	ore:aggregates	Represents the relationship between the Project and its children Collections.

5.2. Collection Model

Usage

A Collection is an aggregation of Data Items (or simply, Items). The Collection maintains metadata that pertain to the aggregated Items and the Collection itself. Collections may be published and cited. Collections can contain multiple sub-collections. A Collection belongs to a single Project.

Model

Property	Cardinality	ORE-ReM representation	Usage
Id	(1,1)	dcterms:identifier ore:similarTo	Generated by the system during ingest (via package ingest) or creation (via the DCS Front End)
Alternate Id	(0,*)	dcterms:identifier	Alternate Id for the Collection
Title	(0,1)	dcterms:title	Short title of the collection
Summary	(0,1)	dcterms:description	A summary description of the Collection's content
Discipline	(0,*)	dcterms:subject	A term of a vocabulary identifying the discipline which the Collection is associated with
Citable Locator	(0,*)	datacons:citableLocator	Unique id which could be used to cite the Collection and its content.
Creator	(0,*)	dcterms:creator	Contain information about the Collection's creator(s)
Creator Name	(0,1)	dcterms:creator/foaf:name	Cardinality is per creator
Creator Phone	(0,*)	dcterms:creator/foaf:phone	Cardinality is per creator
Creator Email	(0,*)	dcterms:creator/foaf:mbox	Cardinality is per creator
Creator Page	(0,*)	dcterms:creator/foaf:page	Cardinality is per creator
Contact		scoro:contact_person	Contain information about the designated point of contact for the Collection
Contact Name	(0,1)	scoro:contact_person/foaf:name	Cardinality is per contact
Contact Phone	(0,*)	scoro:contact_person/foaf:phone	Cardinality is per contact
Contact Email	(0,*)	scoro:contact_person/foaf:mbox	Cardinality is per contact
Publication Date	(0,1)	dcterms:issued	Date on which the Collection is published
Create Date	(1,1)	dcterms:created	Date on which the Collection was first created
Modified Date	(1,1)	dcterms:modified	Date on which the Collection's information is updated
Deposit Date	(1,1)		Date on which the Collection is deposited - system generated upon ingest
Depositor Id	(1,1)		Identifier of the user who deposited the Collection - system generated upon ingest

Relationship name	Cardinality	Targets	ORE-ReM	Usage
hasMember	(0,*)	Collection OR Data Item	ore:aggregates	Represents the relationship between a Collection and its children Collection or its children Data Items
isMemberOf	(1,1)	Collection OR Project	ore:isAggregatedBy dcterms:isPartOf	Represents the relationship between a Collection and its containing Collection or Project
hasMetadata	(0,*)	MetadataFile		Represents the relationship between a Collection and its MetadataFile

5.3. Data Item model

Usage

Data Items (Items) are the unit of deposit, and serve as a container for describing the file or files contained therein. Items must be deposited to a Collection; currently an Item can only be a member of a single Collection, and they cannot be moved between Collections. In this release, Data Items contain multiple data files.

Model

Property	Cardinality	ORE-ReM representation	Usage
Id	(1,1)	dcterms:identifier ore:similarTo	Generated by the system during ingest (via package ingest) or creation (via the DCS Front End)
Alternate Id	(0,*)	dcterms:identifier	Alternate Ids of the Data Item
Name	(0,1)	dcterms:title	A descriptive label of the Data Item
Description	(0,1)	dcterms:description	A summary description of the Data Item's content
Citable Locator	(0,*)	datacons:citableLocator	Unique id which could be used to cite the Data Item and its content.
Content Model	(0,*)	dcterms:conformsTo	Content models to which a given manifestation of a Data Item conforms.
Creator	(0,*)	dcterms:creator	Contain information about the Data Item creator(s)
Creator Name	(0,1)	dcterms:creator/foaf:name	Cardinality is per creator
Creator Phone	(0,*)	dcterms:creator/foaf:phone	Cardinality is per creator
Creator Email	(0,*)	dcterms:creator/foaf:mbox	Cardinality is per creator
Creator Page	(0,*)	dcterms:creator/foaf:page	Cardinality is per creator
Contact		scoro:contact_person	Contain information about the designated point of contact for the Data Item
Contact Name	(0,1)	scoro:contact_person/foaf:name	Cardinality is per contact
Contact Phone	(0,*)	scoro:contact_person/foaf:phone	Cardinality is per contact
Contact Email	(0,*)	scoro:contact_person/foaf:mbox	Cardinality is per contact
Create Date	(1,1)	dcterms:created	Date on which the Data Item was first created
Modified Date	(1,1)	dcterms:modified	Date on which the Data Item's information is updated
Deposit Date	(1,1)		Date on which the Data Item is deposited
Depositor Id	(1,1)		Identifier of the user who deposited the Data Item

Relationship name	Cardinality	Targets	ORE-ReM representation	Usage
hasMember	(0,*)	Data File	ore:aggregates	Represents the relationship between a Data Item and its children Data Files
isMemberOf	(1,1)	Collection	ore:isAggregatedBy dcterms:isPartOf	Represents the relationship between a Data Item and its containing Collection
hasMetadata	(0,*)	MetadataFile		Represents the relationship between a Data Item and its MetadataFile

5.4. Data File model

Usage

Date File, an encapsulation of a bytestream, is the most granular unit of data. Data File has to be contained by a Data Item when being deposited into the system. Data File can only be a member of one Data Item.

Model

Property	Cardinality	ORE-ReM representation	Usage
Id	(1,1)	dcterms:identifier ore:similarTo	Generated by the system during ingest (via package ingest) or creation (via the DCS Front End)
Name	(0,1)	opmv:hasFileName	File system name of the file
Title	(0,1)	dcterms:title	A description label for the File
Description	(0,1)	dcterms:description	A summary description of the File's content
Format	(0,*)	dcterms:format dcterms:conformsTo	Information regarding the format of the File, such as mimetypes, xml schemas, etc.
Size	(0,1)	opmv:hasSize dcterms:extent	Size of the bytestream contain in the File, system generated property
Fixity	(0,1)		Checksum value of the File's bytestream
Create Date	(1,1)	dcterms:created	Date on which the File was first created
Modified Date	(1,1)	dcterms:modified	Date on which the File's information is updated
Deposit Date	(1,1)		Date on which the File is deposited
Depositor Id	(1,1)		Identifier of the user who deposited the File

Relationship name	Cardinality	Targets	ORE-ReM representation	Usage
isMemberOf	(1,1)	Data Item	ore:isAggregatedBy dcterms:isPartOf	Represents the relationship between a Data File and its containing Data Items

5.5. Metadata File model

Usage

Metadata File, an encapsulation of a bytestream, are used to further describe another object (including another Metadata File). Metadata File can only describe ONE other object and can only be deposited as part of the higher level object it (Collection, DataItem) described. Currently a Metadata File cannot describe a Project, but we expect to support this relationship in future releases.

Model

Property	Cardinality	ORE-ReM representation	Usage
Id	(1,1)	dcterms:identifier ore:similarTo	Generated by the system during ingest (via package ingest) or creation (via the DCS Front End)
Name	(0,1)	opmv:hasFileName	File system name of the file
Title	(0,1)	dcterms:title	A description label for the File
Description	(0,1)	dcterms:description	A summary description of the File's content
Format	(0,*)	dcterms:format dcterms:conformsTo	Information regarding the format of the File, such as mimetypes, xml schemas, etc.
Size	(0,1)	opmv:hasSize dcterms:extent	Size of the bytestream contain in the File, system generated property
Fixity	(0,1)		Checksum value of the File's bytestream
Create Date	(1,1)	dcterms:created	Date on which the File was first created
Modified Date	(1,1)	dcterms:modified	Date on which the File's information is updated
Deposit Date	(1,1)		Date on which the File is deposited
Depositor Id	(1,1)		Identifier of the user who deposited the File

Relationship name	Cardinality	Targets	ORE-ReM representation	Usage
isMetadataFor	(1,1)	Collection or Data Item	fedora-rels-ext:isMetadataFor	Represents the relationship between a Data File and its Metadata Files

7. DCS Package Tools

DCS Package Tools are a set of desktop based, standalone applications which provide data packaging and data package validation capabilities. The tools set consist of DCS Package Tool GUI and DCS Package Validation Command Line apps. The following sections covers each of the tool in more details.

7.1. Package Tool GUI

7.1.1. Administration Guide

Prerequisites

Package Tool GUI is a standalone, platform independent desktop application and can run outside of the presence of other DCS components. However, it requires Java 7 to execute.

Installation

1. Obtain a distribution file for Package Tool GUI from Downloads section.
2. Unpack the file into a single directory (extracted directory)
3. Verify that there are a executable `.jar` file and a `lib` directory under the extracted directory

Running & exiting Package Tool GUI

1. To run the Package Tool GUI, execute the following command **from the extracted directory**

   ```
   java -jar dcs-package-tool-gui-<version>-jfx.jar
   ```

   OR double click on the executable `jar`.

2. To exit/close the application, close the main GUI windows

Verification

Go through the workflow as described in the Package Tool GUI User Guide section to verify that the installation works. The expected outputs when using the tool is a Package Description file in `json` format and a package file in the format specified in the last step of the workflow.

Configuration

The Package Tool GUI is configured via the use of `.properties` files contained in the executable jar. To modify the default configuration, the executable jar would need to be unpacked, relevant properties files updated, another executable jar produced to include the updated files.

General configuration

Configuration relating to the operation of the application as a whole is set in `config_default.properties` file under `bundles/` directory. By default, the application is configured with path to the default package generation parameters file, path to the `.properties` file which maps object types and property to human-readable text (as mentioned below), and a list of characters deemed to be illegal in naming a package.

Texts configuration

Texts used on the GUI are configured in the following files under `bundles/` directory:

File name	Content

`bundles/dcsbopropertylabels.properties`	Maps the ontology property and object type code to human readable text to be used on the GUI
`bundles/error.properties`	Configures error messages used in the GUI
`bundles/help.properties`	Specifies the location where files containing help text used in the GUI can be found and loaded.
`bundles/label.properties`	Specifies the text used on buttons and labels in the GUI
`bundles/messages.properties`	Specifies messages (regarding both failing and successful conditions) used in the GUI
`bundles/revision.properties`	Specifies the code revision number, application build number and time stamp of when the build occurs (this properties file should not be edited.
`*.help`	Configures the text in help pop ups.

Default package generation parameters

Package generation process operates on a set of parameters, the some of which (less frequently changed ones) are given default values in `defaultGenerationParams` file under the top level directory of the extracted `jar`. For more details about the package generation parameters, see Package Generation Parameters page.

7.1.2. User Guide

The Package Tool GUI User Guide is targeted at users of the DCS Package Tool GUI client application.

7.1.2.1. Introduction

The DCS Package Tools Graphical User Interface (GUI) is a client application designed to be run locally on a data producer's machine. The purpose is to allow the data creator to easily create ingest-ready packages for an DCS Archive instance installed on another server. It does this by allowing users to use file-directory semantics to describe hierarchical relationships amongst data objects (Collections, Data Items) while letting users describes their data interactively. The package created can be anything from an entire project, to a collection within an existing project.

This document will assume a working installation of the GUI. (Installation is covered in the Package Tool GUI Administration Guide.) A DCS Archive installation is not required to be accessible while using this application, as the package is not ingested directly into the system, but rather the GUI outputs a package file which can be used to ingest into the DCS Archive at another time.

7.1.2.2. Terminology and Data Model

Terminology

Here you can find definitions and explanations for some of the terms used elsewhere in this documentation and in the application.

Package Artifact (Artifact) - A first-order entity included within a package with a distinct identity relative to the package. Each artifact is generated to represent a file-system entity to be included in the package. A package artifact should contain enough descriptive information (artifact type, artifact name, created date, etc) about a file-system entity so that the entity can be transformed into a Business Object.

Package Description - An aggregation of Package Artifacts which as a whole describes a file system directory structure that the user selects

during package generation. The package description itself is stored in memory while using the GUI - changes are not immediately stored to disk until the user hits the Save buttons.

Package Description Document - A serialization of the Package Description to disk when using the GUI. This can be loaded later to resume editing a Package Description.

Data Model

Each Artifact has a set of properties associated it. Many of these properties are optional. In the GUI itself, some properties are marked with a *, to indicate they are the minimally recommended properties for the artifact. However, in order to successfully ingest a package into a DCS Frontend instance, there are more required fields; without them, the ingest may fail. In the lists below, any property marked with ** in front are required for package ingest in the DCS Frontend. Properties marked with a + can have multiple values (see Multi-Valued Properties for information)

PROJECT

Note that currently the DCS Frontenddoes not ingest new projects, but only ingests collections, etc, to active projects. Also, when a new project is created, the project ID will be created anew. Please see the 5. DCS Business Object Ontology for information on the Project artifact ontology.

- **ID
- **Start Date
- **Description
- **Name
- **End Date
- **Funding Entity
- **Number (IE, Award numbers) +
- Principle Investigator +
- Storage Allocated
- Storage Used
- Publisher

COLLECTION

- **Created Date
- **Modified Date
- **Title
- **Description
- **Creator (with fields for **Name, Phone +, Web Page +, and Email +)+
- Publication Date
- Publisher
- Alternate Id +
- Contact Info (with fields for **Name, Phone +, and Email +) +
- Citable Locator
- Discipline +

DATA ITEM

- **Created Date
- **Modified Date
- **Name
- Description
- Alternate Id +
- Contact Info (with fields for Name, Phone +, and Email +) +
- Citable Locator +
- Content Model +
- Creator (with fields for **Name, Phone +, Web Page +, and Email +) +

DATA FILE

- **Created Date
- **Modified Date
- **File Name
- **Size (System generated, cannot be modified, read only)
- Format
- Title
- Description

METADATA FILE

- **Created Date
- **Modified Date
- **File Name
- **Size (System generated, cannot be modified, read only)
- Format

- Title
- Description

7.1.2.3. Workflow

The following sections provide an overview and walk-through of creating a package file.

Screen 1 - Create New Package

Screen 2 - Define File Relationships and Artifact Properties

- Modifying Artifact Properties and Relationships
  - Artifact Properties - General
  - Artifact Properties - Creator
  - Artifact Properties - Relationships
  - Artifact Properties - Inheritance

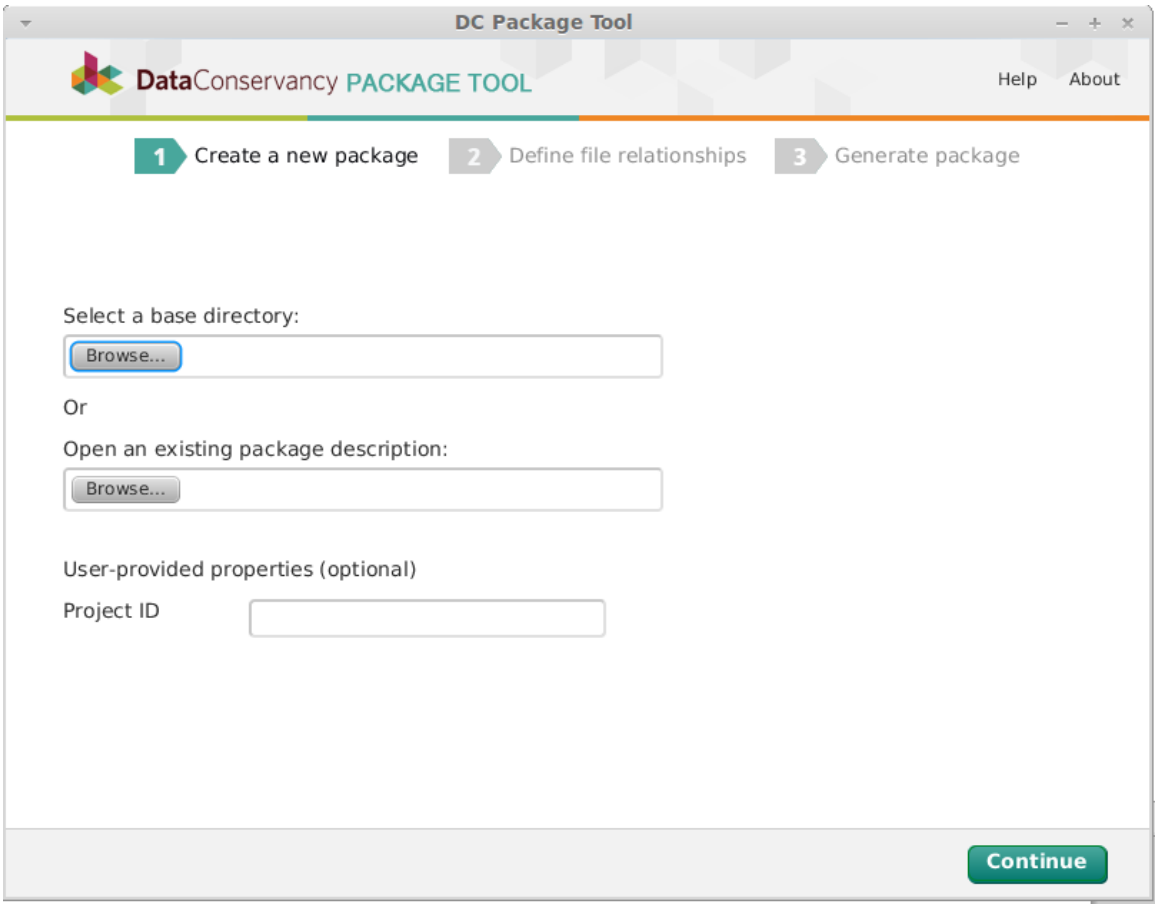
Screen 3 - Generate Package File

- Package Generation Parameters

Screen 1 - Create New Package

When the application initially opens, you will have the opportunity to create a new package by selecting the base directory that package files are in. Also, you can open an existing package description, for situations where you saved a package description prior to finishing the changes.

The initial screen looks as follows:



To proceed, one of the top two boxes (but not both) must be filled.

Select a base directory - If you wish to begin a new package, select the *Browse...* button in this field, and navigate to the root directory

containing the files you wish to include in the package.

Open an existing package description - If you wish to continue working from a previously-saved package description document, select the *Browse...* button in this field, and navigate to the file you had previously saved.

You can also specify some user-provided properties. These are optional to create the package.

Project ID - Required if the user is creating a package to be ingested by the DCS Archive. This is the Business ID of an existing Project in the DCS. If you don't know the Business ID of the project browse to the project in the DCS Frontend. On the project details page click the Object ID Export link this will download a small xml file that contains all the Business IDs in your project. You should get a file that looks something like this:

```
<bo>
   <id>
      http://localhost:80/dcs-ui/project/1
   </id>
   <name>test project</name>
   <type>Project</type>
   <depositStatus>DEPOSITED</depositStatus>
</bo>
```

The value http://localhost:80/dcs-ui/project/1 is the Business ID of test project.

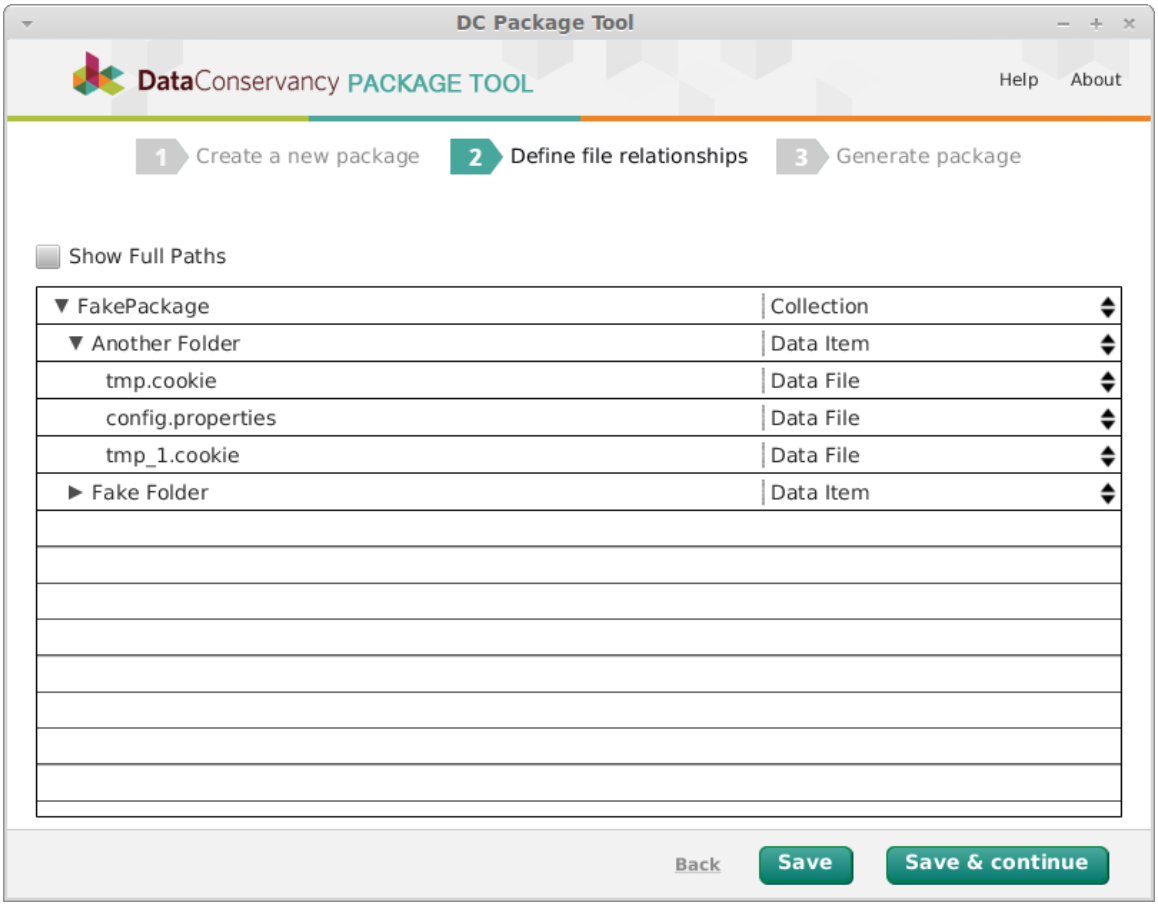
If creating packages not intended to be put into the DCS Archive, this field is optional.

When you have filled in the relevant fields, simply press the Continue button to proceed to Screen 2, where you can add and/or modify description about your data content.

Screen 2 - Define File Relationships and Artifact Properties

Once the package directory has been selected (or an existing package description file is loaded), the corresponding Package Description is presented on the second screen of the workflow. User can modify the various artifacts within the Package Description with metadata and specify additional relationships to create a desired Package Description. File system directory structures are hierarchical by nature; therefore, they transform fittingly to the hierarchical graph of the DCS Business Objects. By default, the directory structure will define some of the basic artifact types and relationships - folders may represent projects, collections, or data items; files may represent data files and metadata files; a file within a directory may be considered a Data File within a Data item, etc.

The loaded Package Description is displayed in a ***tree*** view, with each node represent a package artifact:



Show Full Paths - If this checkbox is checked, all entries in the tree will have the full path to the related file/folder in the file system. These filenames are URL-encoded (so spaces will show as %20, for example). If unchecked, only the un-encoded tail portion of the path (IE, the actual file or folder name) will be displayed in the tree, though you can still determine relative paths by the tree structure itself.

The tree itself contains a series of rows. Each row represents an artifact of the package. Each row has the following features:

- The arrow on the far left allows you to expand or collapse the tree at this node to show the artifact's "children". If there is no arrow, then the artifact has no child artifacts.
- The name of the artifact (or full path, if the Show Full Paths box is checked), corresponding to the folder or file name on the system.
- The type of the artifact. This will be either Project, Collection, Data Item, Data File, or Metadata File.
- The double-arrow on the right, when clicked, will pop up a context menu for the artifact. The context menu will give options to change the artifact's type, as well as to modify the artifact's properties. The types that you are allowed to change to depend on the artifact and tree structure; some artifacts cannot be changed

One of the options in the double-arrow menu is to edit the artifact's properties. See the following for information:

Modifying Artifact Properties and Relationships

- Artifact Properties - General
- Artifact Properties - Creator
- Artifact Properties - Relationships
- Artifact Properties - Inheritance

At the bottom are a series of buttons and links.

Back - Pressing this will return you to Screen 1 of the application.
Save - Pressing this will allow you to save the artifact tree to a package description file. This file can be used later load from Screen 1. After saving, you can continue working on the tree.
Save & Continue - Pressing this functions similar to Save, except after saving you will be taken to Screen 3 of the application, where you can complete the package generation process.

Modifying Artifact Properties and Relationships

The Properties dialog allows you to:

- modify properties for a given artifact
- add new relationships between a particular artifact and other things (including external entities)

Note that different artifacts may have different properties and relationships allowed. If you change the artifact type, any properties that do not apply to the new type will be lost when the package description file is saved.

Multi-Valued Properties

Many of the properties on the various tabs have a (+) next to them. These properties are allowed to have multiple values. The button is disabled, however, unless all previous fields have values. Pressing the button will cause a new, blank field for that property to be added. Note for these properties, the values are not stored in any particular order, so when the dialog is re-opened, they may not display in the same order they were originally entered.

If you wish to remove a value from these properties, simply erase the value in the field (leaving it blank). When the *Apply* button is pressed, all empty values will be removed from the property.

Grouped Properties

Some of the properties you can modify are grouped into sections, such as *Contact Info*, *Creator*, and *Relationships*. Each of these group properties have an *Add* button at the bottom of the last section, to allow a new property group to be added. This button is disabled unless each section has at least one valid value filled in for one of the fields. If any section is completely empty, you will not be able to create a new section.

To remove an existing section, simply empty all fields for that section (including all fields in a multi-valued property in the group). When *Apply* is pressed, any empty group will be removed from the artifact.

Bottom Buttons

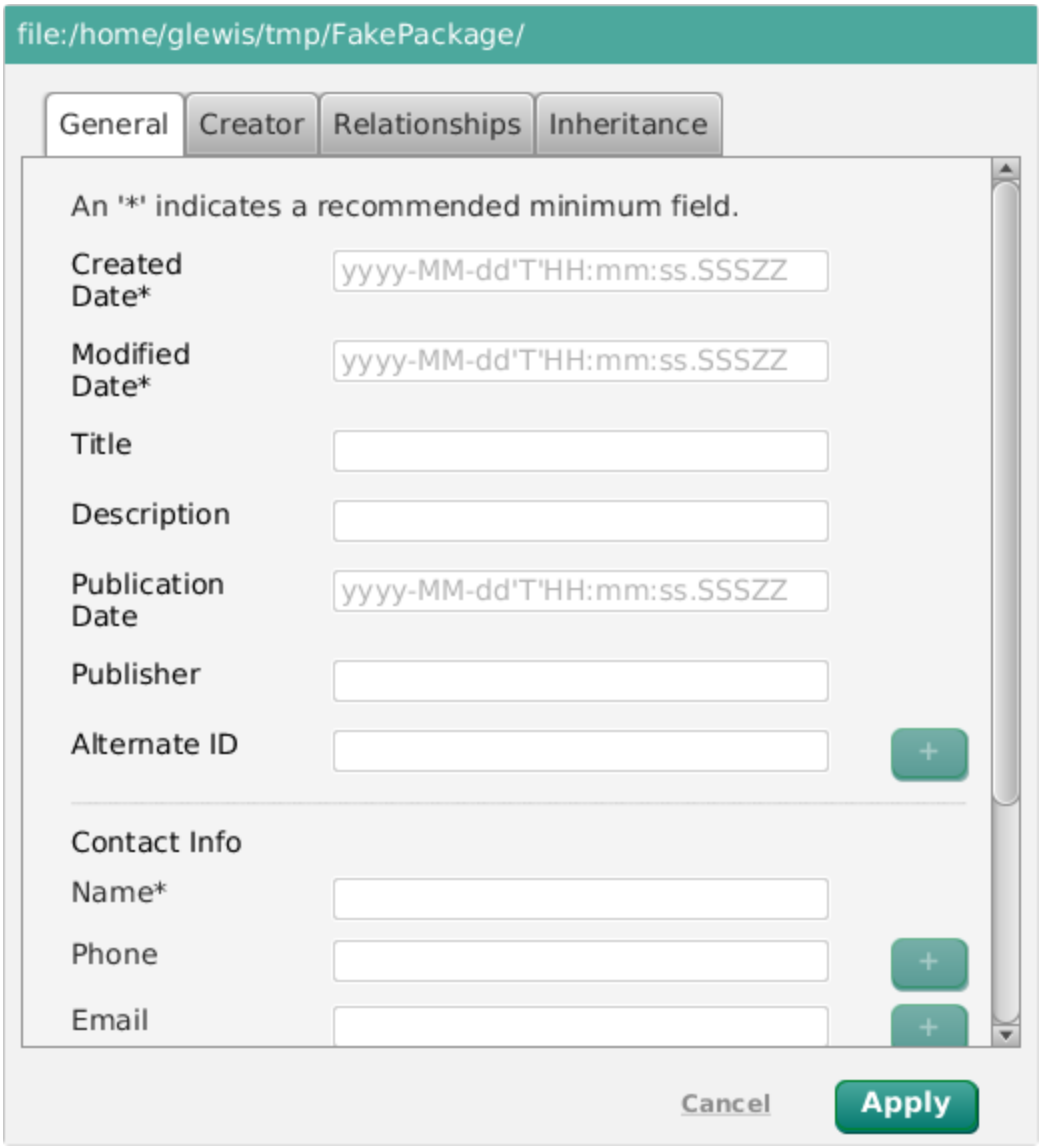
At the bottom of the dialog are two options. Both will close the dialog window.

Cancel - Press this if you do not wish to save the changes you have made to the properties, etc.

Apply - Save the changes made to the properties for the artifact. **NOTE:** The changes saved here are NOT saved to disk, but simply to the Package Description model stored in memory. To save the changes to disk, you will need to use the *Save* or *Save & Continue* buttons on Screen 2.

Artifact Properties - General

The general tab looks as follows:



As mentioned, some of these fields may be different depending on the type of artifact you are modifying properties for.

Recommended Properties

Any property marked with an asterisk (*) is a minimally recommended field. If not supplied, either a default value will be used, or the property will not be saved (depending on the property).

NOTE: In order to create a DCS Frontend ingestable data package, a number of properties have to be populated with values. For details about which properties are required for ingest, see Terminology and Data Model page.

Contact Info

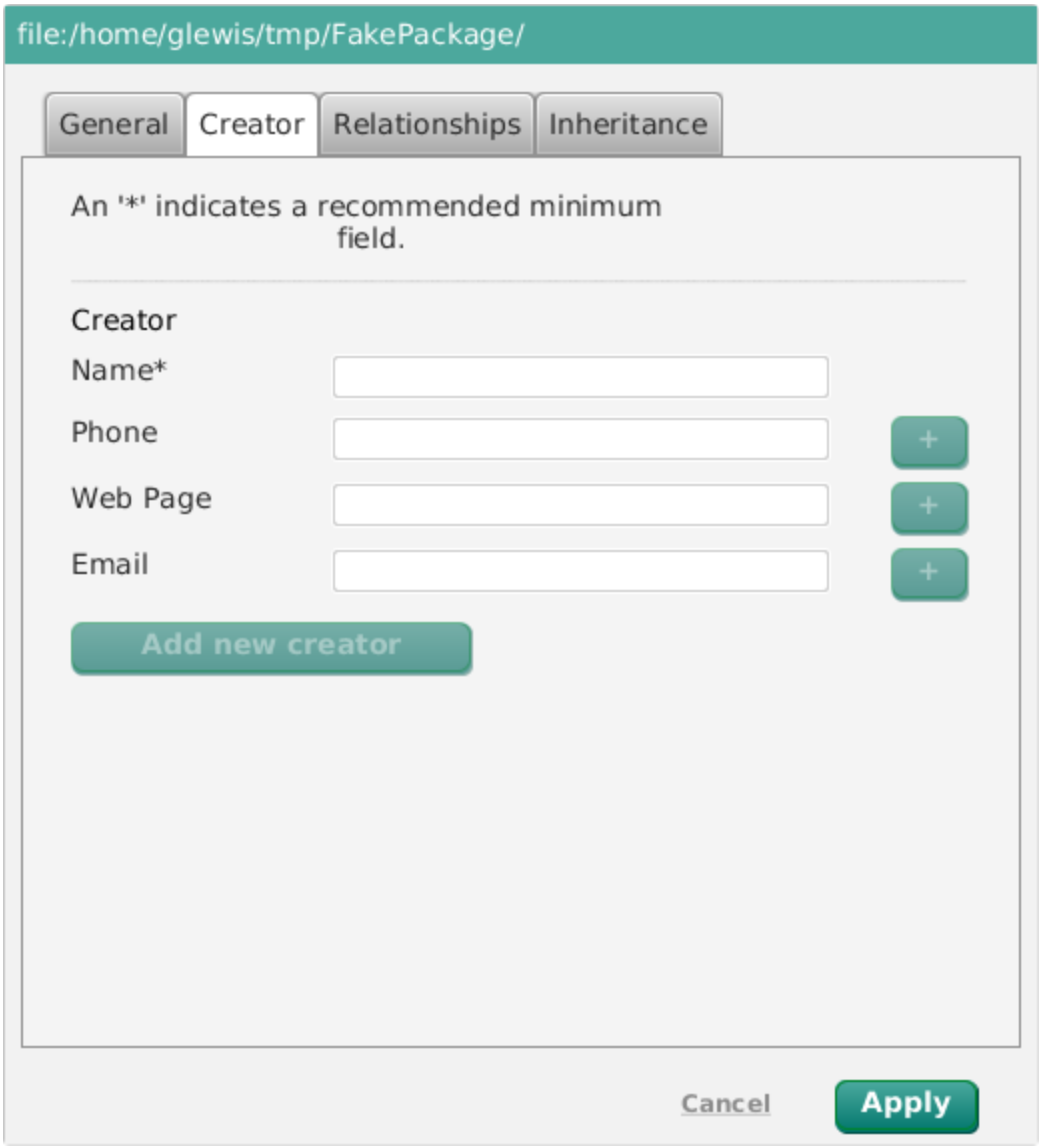
The *Contact Info* property is actually a complex Grouped Property. Each Contact Info has a Name, as well as zero or more *Phone* and *Email* sub-properties. You can add more than one *Contact Info* by pressing the *Add New Contact Info*, and you can add more email and phone sub-properties by pressing the (+) next to the related field. For more information, see Grouped Properties for details on adding/removing entire *Contact Info* sections, and Multi-Valued Properties for adding/removing values from the phone and email sub-properties.

Available Properties

For details on which properties are available for the different artifact types, please see the Terminology and Data Model page.

Artifact Properties - Creator

The *Creator* page looks as follows:

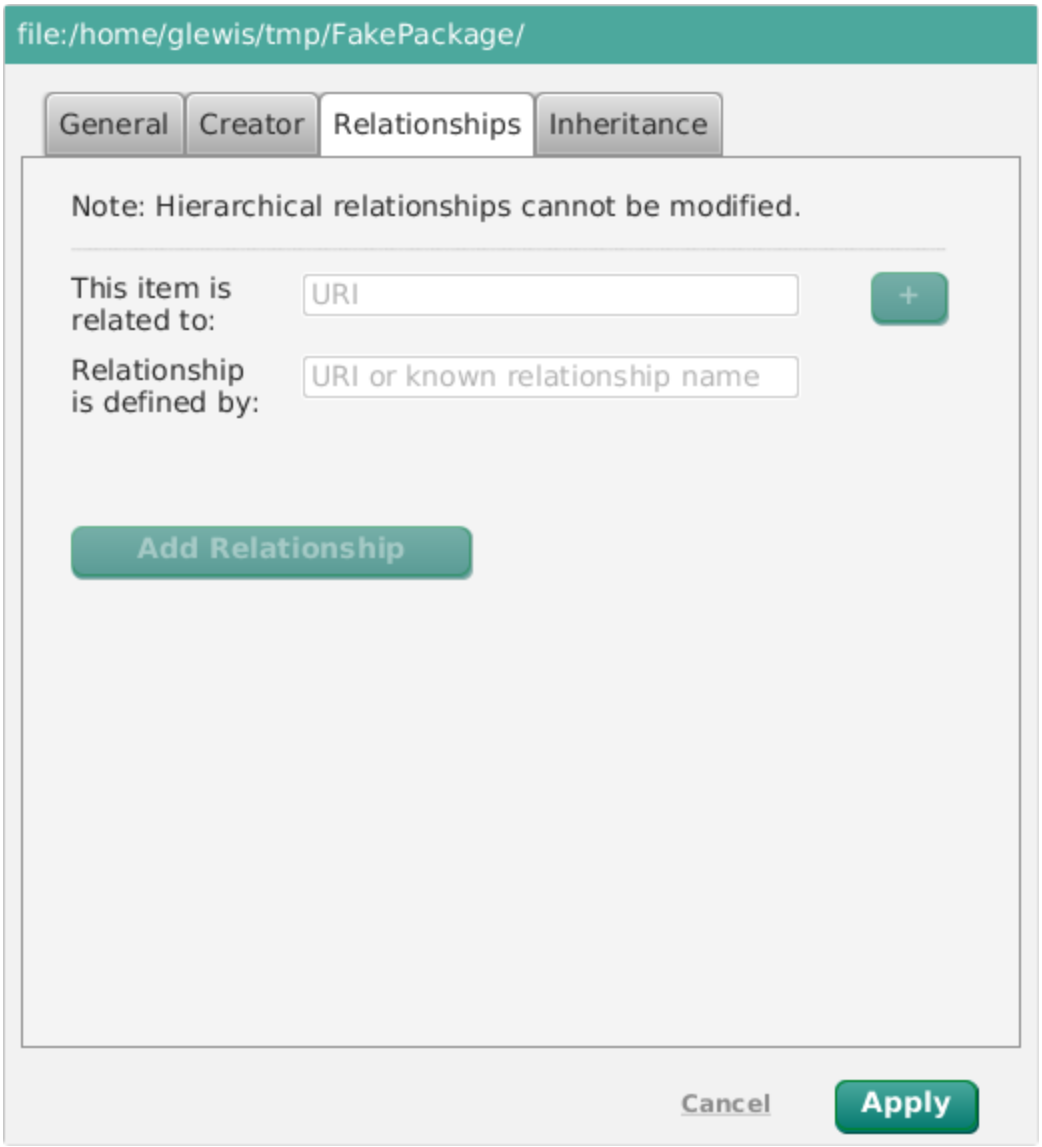


Each *Creator* is a complex Grouped Property. Each creator is required to have *Name*, and can optionally have zero or more *Phone*, *Web Page*, and *Email* sub-properties. A new creator can be added by pressing the *Add new creator* button. New phone, web page, and email values can be added by pressing the (+) next to the related field. Please see the Grouped Properties section for details on adding/removing creators, and Multi-Valued Properties for details on adding/removing phone, web page, and email sub-properties..

Note that *Creators* are very similar to how *Contact Info* properties work from the *General* screen, but have an additional field - *Web Page*.

Artifact Properties - Relationships

This is what the *Relationships* tab may look like:



Each *Relationship* consists of two components - an item (or items) that this artifact is related to, and the definition of the relationship. These function similarly to the complex Grouped Properties such as *Contact Info* from the *General* tab or *Creator* from the *Creator* tab. You can add multiple relationships by using the *Add Relationship* button - see the Grouped Properties section for details on adding and removing relationship groups

Note that by default, many artifacts include a relationship based on the tree hierarchy - all children artifacts have the "isMemberOf" relationship to its parent artifact. These hierarchical relationships cannot be removed or modified.

The two component properties of a relationship are described here:

This item is related to - You can specify one or more "things" (called targets) which relate to this artifact, as long as the targets are specified by a valid URI string. Adding new targets is done by pressing the (+) button next to the field. Please see the Multi-Valued Properties section for more details on adding/removing values for this component.

Relationship is defined by - Defines how this artifact relates to the target(s). Any valid URI can be used, in the form scheme:scheme_specific_parts. For instance, relationship:mine would be valid, as would relationship:mine/is/awesome.

There are some hierarchical relationships pre-defined by the system and are automatically generated based on the Package Artifacts tree structure. These relationships' definition are not in the form of URI, but they cannot be manually added. For reference, they are:

- isMemberOf
- hasMember
- isMetadataFor
- hasMetadata

Visit the Ontology section of this document for more details on what each of these relationships means.

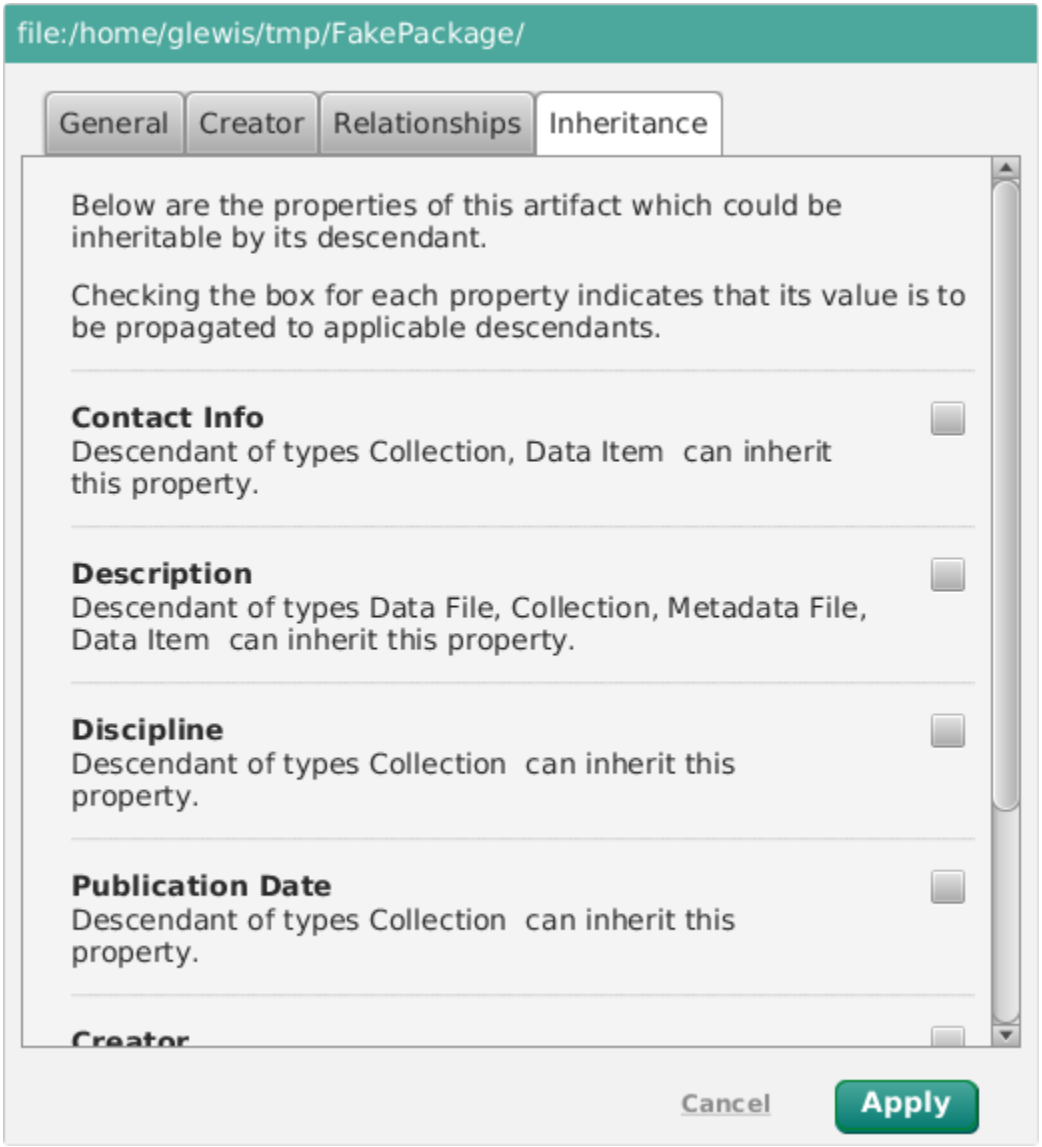
Note: Metadata File typed of artifacts cannot be "isMetadataFor" artifacts of the same type or artifacts of Data File type. While the packaging tool

GUI will not prevent this, adding these relationships will cause a failure when the package is ingested into the DCS Archive. More precisely, a node in the package artifact tree cannot be a "child" of two different node on the same tree, and the above relationships are considered hierarchical and thus imply a parent-child relationship.

Custom relationships do not have such restrictions, as they are not considered hierarchical relationships.

Artifact Properties - Inheritance

The *Inheritance* tab looks as follows:



This tab allows you to more quickly propagate properties to an artifact's children, without having to re-enter the same data manually for each one. When *Apply* is pressed, a checked checkbox on this tab will specify that its associated property's value is to be be inherited by descendantss artifacts downward through the tree.

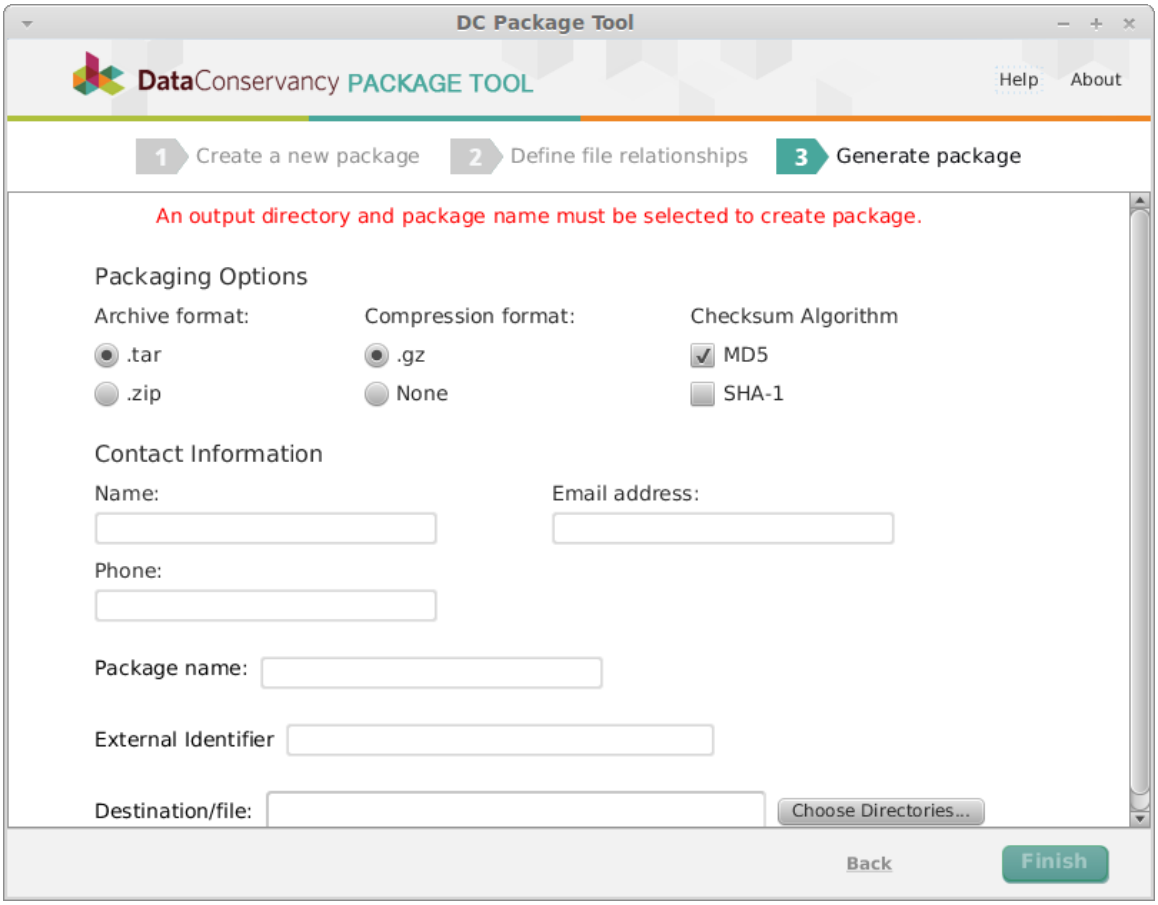
Each property will tell you what types of artifacts will inherit the specified properties. For instance, Collection and Data Item can have *Contact Infos* and *Creators*, while Data File and Metadata File artifacts cannot. If you choose to propagate a Collection's *Contact Info*, for instance, all sub-Collections and Data Items will automatically have the *Contact Info* added, but Data File and Metadata File children will not.

Depending on the artifact, different options will appear on this tab. For leaf-node artifact types (Data File and Metadata File), there will be no options available.

Screen 3 - Generate Package File

Once the artifact properties and relationships have been defined, you can proceed to generate the actual package file. On the final screen, you have the opportunity to enter some package-level information and decide where to store it and how the package file should be bundled.

The third screen looks as follows:



Note that initially, a red warning will appear to let you know that a *Package Name* and a destination directory must be selected. The *Finish* button will be disabled until both of these fields are entered. Additionally, this red warning may change after attempting to save a package if there is an error generating the package file.

The various fields are described below:

Packaging Options

Archive Format - This describes what type of file archive you wish to use, tar or zip.

Compression Format - For tar files, specifies whether you wish to compress it with gzip compression or not. If using the zip Archive format, this option is always None.

Checksum Algorithm - If checked, a pair of checksum files will be generated for each algorithm and placed in the package. These checksum files are used to verify that the files are unchanged since when they are added to the package. You can select either or both algorithms. If none

Contact Information

None of the fields in this section are required. The information in this section is used to describe the contact information of the data creator who is creating the package file. This information will be put in the package description files included within the package.

Name - The name of the person to contact regarding the package file.

Phone - A phone number that the contact person could be reached at.

Email - An email address the contact person can be reached at

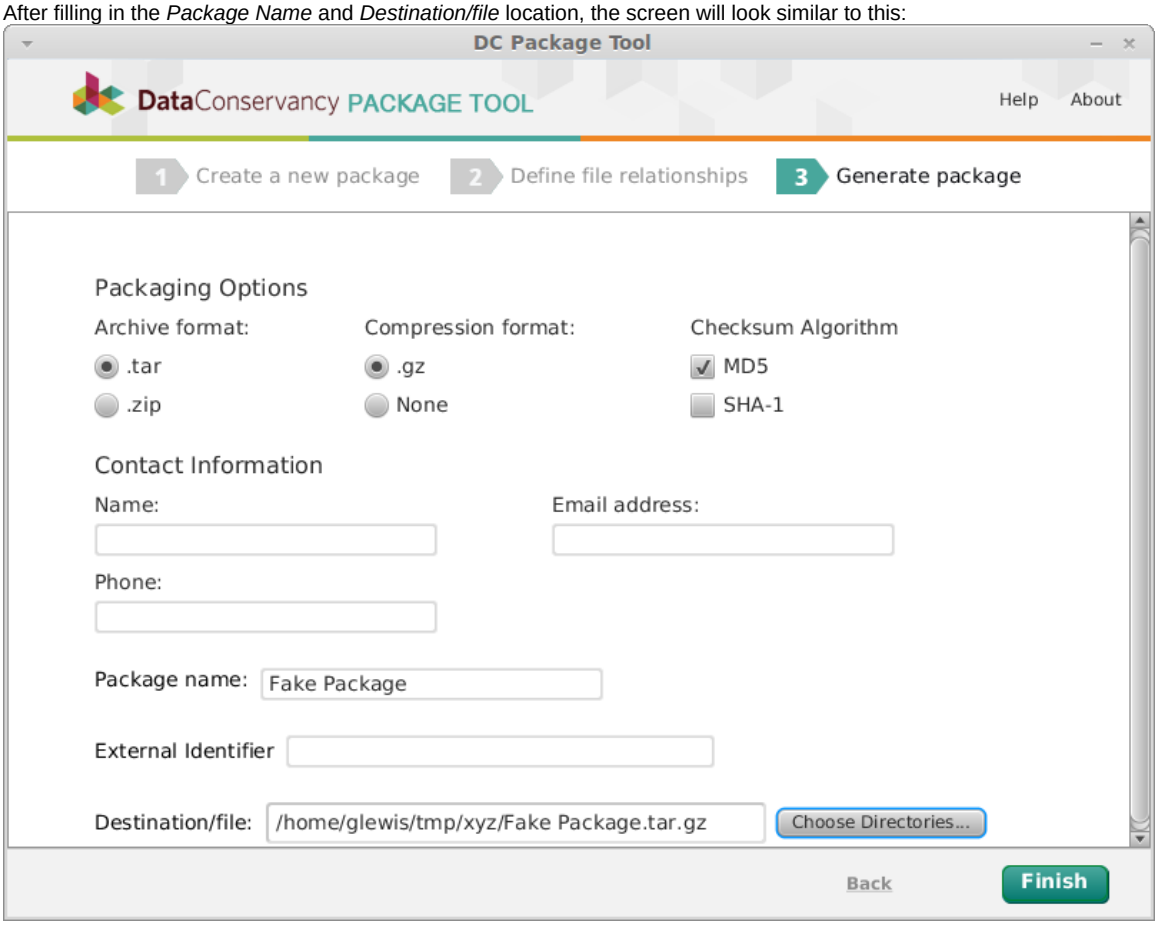
Other

Package Name - The name of the package. This will be used inside the package file as the name of the root level folder for the package, as well as the core name of the package file itself. This field is required.

External Identifier - This information is used to populate `External-Identifier` property in `bagit-info.txt` file in the final BagIt package file. This is a required field for ingestion into DCS Frontend.

Destination/file - You cannot edit the box manually. If you press the *Choose Directories*... button, you will be presented with a file selection window; navigate to the directory you wish to save the package to, and hit "Open". If a *Package Name* has also been specified, the *Destination/file* field will show the full path to where the package file will be saved. This will also cause the red warning at the top to disappear. However, if either the directory or Package Name are NOT defined, this box will be blank and the red warning will be displayed.

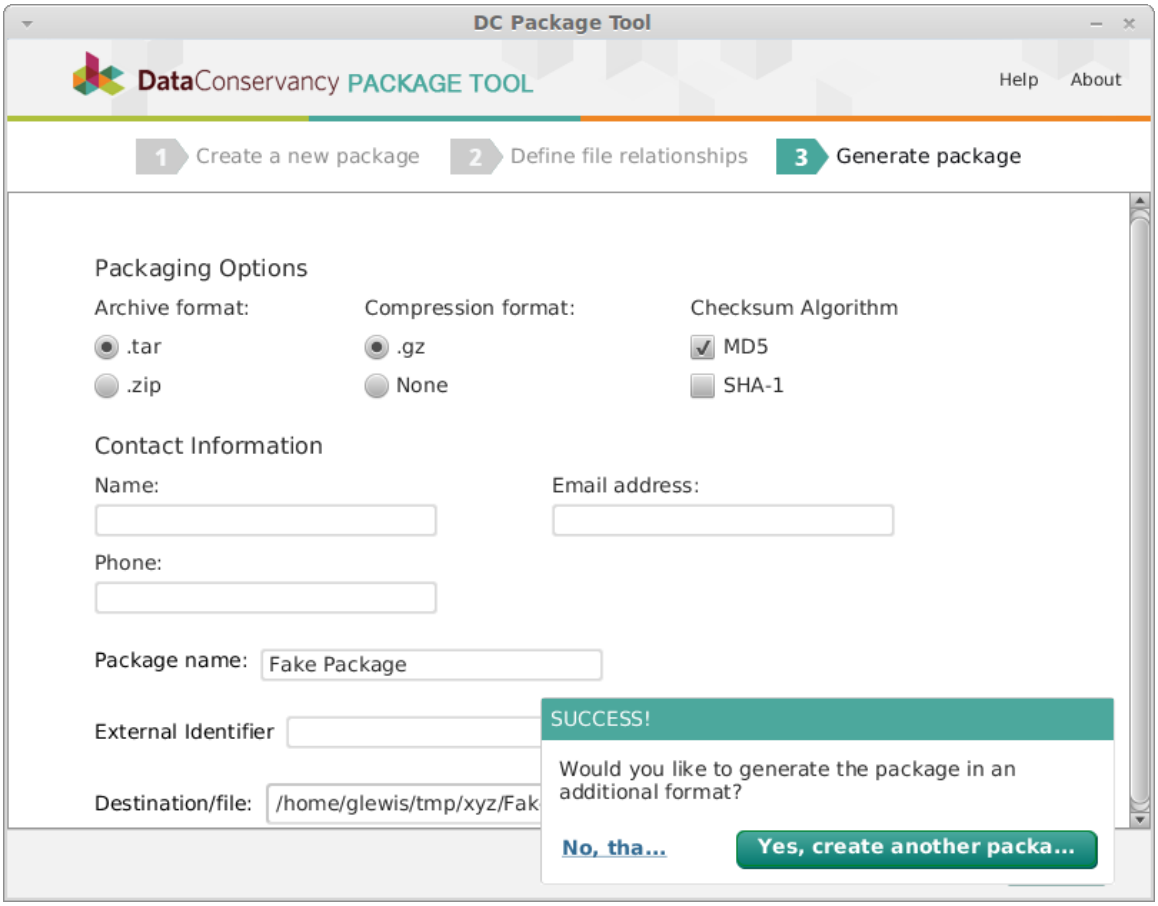
After filling in the *Package Name* and *Destination/file* location, the screen will look similar to this:



Pressing the *Back* button will return you to Screen 2 to allow you to continue modifying properties and relationships.

Pressing finish will cause the package file to be saved at the location specified in the *Destination/file* directory. If the file already exists, you will be prompted as to whether you want to overwrite the existing file. If you do, this action cannot be reversed. If you do not wish to overwrite the file, either change the destination directory, change the package name, or move the existing file to another location before saving.

If any errors occur while saving the file, a red warning message will let you know what went wrong. If no errors occur, the package file will be saved, and you will be shown a small dialog to see what you want to do next, as shown here:



If you press *No, thanks*, you will be taken back to Screen 1 to begin the process of creating a new package.

If you press *Yes, create another package*, you will remain on this screen. This would be beneficial if you wish to save the same package in different formats (for example, a zip format and a tar.gz format), or if you wish to save the package in multiple locations.

Package Generation Parameters

Required parameters

In order to produce the package a minimal set parameters are required. Some of these parameters value requires user's input each time a package is generated. Some others (less frequently changed) are configured in defaultGenerationParams file:

Required parameter name	Value comes from	Usage
Package-Name	User's input via the GUI	Used to create the resulting data package file
Content-Root-Location	User's input via the GUI	Used to generate information in the package
Contact-Name	User's input via the GUI	Part of BagIt metadata
Contact-Email	User's input via the GUI	Part of BagIt metadata
Contact-Phone	User's input via the GUI	Part of BagIt metadata
Package-Format-Id	Set in defaultGenerationParams file	Used to generate data package in proper format
BagIt-Profile-Identifier	Set in defaultGenerationParams file	Used to generate data package in proper format

Optional parameters

Other parameters can be provided to add more metadata to a package. Some of these parameters are generated by the Package Tools application, others could be specified in the defaultGenerationParams file. The values provided are used in a few different way as specified below:

Parameter name	Value comes from	Usage

`Checksum-Algs`	User's input	To determine which checksum algorithm is used in creating the BagIt package's manifest files
`Compression-Format`	User's input	To determine how the BagIt package can be compressed
`Archiving-Format`	User's input	To determine how the BagIt package can be archive
`External-Identifier`	User's input	Part of BagIt metadata
`BagIt-Version`	System supplied	Part of BagIt metadata
`Tag-File-Character-Encoding`	System supplied	Part of BagIt metadata
`Bagging-Date`	System supplied	Part of BagIt metadata
`PKG-BAG-DIR`	System supplied	Part of BagIt metadata
`Bag-Size`	System supplied	Part of BagIt metadata
`Payload-Oxum`	System supplied	Part of BagIt metadata
`Source-Organization`	Could be set in defaultGenerationParams	Part of BagIt metadata
`Organization-Address`	Could be set in defaultGenerationParams	Part of BagIt metadata
`External-Description`	Could be set in defaultGenerationParams	Part of BagIt metadata
`Bag-Group-Identifier`	Could be set in defaultGenerationParams	Part of BagIt metadata
`Bag-Count`	Could be set in defaultGenerationParams	Part of BagIt metadata
`Internal-Sender-Identifier`	Could be set in defaultGenerationParams	Part of BagIt metadata
`Internal-Sender-Description`	Could be set in defaultGenerationParams	Part of BagIt metadata

7.2. Package Validation Command Line

Installation

1. Download either the .zip or or .tar.gz packaging tools command line distribution
2. Unzip or untar the archive into an appropriate place

The packaging tool distribution is simply a .zip or .tar.gz archive containing java libraries, and scripts to launch the tool on Windows and Unix-like platforms. The significant components within the archive are:

- `bin/` directory: This contains scripts for running the validation application
- `lib/` directory: This contains java libraries
- `bin/pkg-validate.bat`: This is a batch file for running the cli on Windows
- `bin/pkg-validate.sh`: This is a script for running the cli on Unix-like platforms (e.g. Linux, Mac).

If installing the cli tools for system-wide use on Linux, it makes sense to unzip the packaging tool into a directory such as `/usr/local`, then create a symbolic link to `pkg-validate.bat` from `/usr/local/bin`, so that the script is within users' default path.

PackageValidationApp

This application validates package descriptions. The shell scripts wrapping this application are pkg-validate.sh and pkg-validate.bat, for Linux and Windows environments, respectively. Currently, this application verifies that the basic Package Description structure is free of error.

The app is invoked as (using the linux example):

```
pkg-validate.sh [infiles]
```

. where `infile` is the location of the package description file(s) (or `'-'` for `stdin`).

Optional command line arguments are

- `-h (-help, --help)` which prints a help message
- `-v (-version, --version)` which prints version information