Oracle® Unversal Records Management Adapter Services Reference Guide

10*g* Release 3 (10.1.3.3.0)

March 2007

Adapter Services Reference Guide, 10*g* Release 3 (10.1.3.3.0)
Copyright © 2007, Oracle. All rights reserved.

Contributing Authors: Eric Raney

Contributors: Darin Anderson, Stuart Edeal, Maple Jiao

Table of Contents

Chapter
1

Introduction

Overview

This section contains the following topics:

About This Guide

Universal Records Management (URM) allows organizations to manage their records and retention policies, disposition processes, and litigation or audit holds in a central repository known as a URM Server. They can then apply those policies, dispositions, and holds to content stored in multiple repositories through URM adapters. The repositories can be any server or application that holds content whose retention is to be controlled.

The repository might hold records that need to be preserved for a retention period, specified in a corporate retention schedule, and then destroyed according to a corporate disposition process. The records can be preserved in place if the repository has the ability to ensure that the record will remain unalterable during the retention period. The repository also needs to be able to purge the records (overwrite the files on the disk multiple times) at the end of the retention period. Adapters associated with repositories that cannot manage records as described might have to move the records to URM for proper preservation and disposition.

Repositories might also hold content that is not considered a business record; where there is no requirement that the content be preserved for the retention period. However, it might be desirable to ensure that the content is deleted when the cost or risk of maintaining it outweighs its value to the organization, as defined in the corporate retention policies. At that time, the content would be disposed of according to the disposition processes stored within URM.

With both records and non-records, there is an obligation to ensure that any material subject to a litigation or audit hold (or "freeze") is not deleted, either by a user or as part of a disposition process. The adapters enable URM to ensure that does not happen.

Typically, there is one adapter for each repository. The adapter is positioned close to the repository to ensure that the adapter performs its functions without consuming too much processing or network bandwidth. Adapters can be written as an application extension (e.g., a dynamic load library, or DLL) of the repository.

This guide describes how an adapter interacts with URM while performing the following functions:

1. **Adapter Registration**—The adapter must register itself to URM so that URM knows about the repository and is ready to manage content stored within that repository. For details, see Registering Adapters (page 2-2). The following additional tasks might be necessary:
    a. You might want to update external field mappings after registering the adapter. For details, see Updating External Field Mappings (page 2-7).
    b. You might want to add external custom fields after registering the adapter. For details, see Adding External Custom Fields (page 2-9).
    c. You might want to update external custom fields after registering the adapter. For details, see Updating External Custom Fields (page 2-13).
    d. You might want activate the extra metadata field after registering the adapter. For details, see Setting Up the Extra Metadata Field (page 2-16).
2. **Repository Monitoring**—The adapter can monitor the repository and inform URM of any changes in the repository that affect disposition processes or litigation or audit holds. The adapter can inform URM when the following events occur:
    a. Items that need to be managed externally by URM are added to the repository. For details, see Declaring Items to URM (External Checkin) (page 3-2).
    b. Managed items in the repository have their metadata or properties changed, either by a user or by the repository. For details, see Updating Items in URM (page 3-7).
    c. Items in the repository either need to be managed externally by URM or have their metadata updated in URM, but the adapter does not know if the items exist already in URM. In this case, the adapter can provide URM with item metadata and allow URM to determine whether the items should be declared or updated; if

the items do not exist URM performs a declaration, and if the items do exist URM performs an update. For details, see Declaring or Updating Items in URM (page 3-12).

d. Managed items are deleted from the repository. For details, see Deleting Items from URM (page 3-17).

e. Items that need to be managed internally by URM are added to the repository. For details, see Checking Items into URM (Internal Checkin) (page 3-22).

f. Items that are being managed externally by URM need to be moved to URM and managed internally. For details, see Transferring Items to URM (page 3-25).

3. **Performing URM Tasks**—The adapter can ask URM periodically for tasks that need to be performed within the repository. Those tasks enable URM to abide by the corporate retention policies and disposition processes. The adapter can perform the following tasks:

   a. URM might want the adapter to perform a search within the repository and provide a list of items matching the search criteria. For details, see Performing Federated Searches (page 4-1).

   a. As part of a scheduled disposition process, URM might want the adapter to purge (or dispose by some other means) items that are held within the repository. As part of this process the adapter might create archive zip files, which it can upload to URM for storage. For details, see Performing Dispositions (page 4-5).

   b. When a litigation or audit hold/freeze applies to content stored within a repository, URM might want the adapter to retrieve a list of affected items and preserve them to ensure that they are not edited or destroyed. For details, see Performing Holds/Freezes (page 4-10).

   c. When a litigation or audit hold/freeze is removed, URM might want the adapter to stop preserving the items. For details, see Removing Holds/Freezes (page 4-14).

4. **Querying URM**—The adapter can retrieve the following information from URM:

   a. The adapter can retrieve a retention schedule from URM. It can retrieve either the entire retention schedule or only part of the schedule for viewing. For details, see Requesting a Retention Schedule (page 5-1).

   b. The adapter can request URM metadata for an item. For details, see Requesting URM Metadata for an Item (page 5-4).

   c. The adapter can retrieve the lifecycle for a particular item. For details, see Requesting the Lifecycle for an Item (page 5-5).

5. **Managing Communications**—The adapter can perform the following tasks related to communication between the adapter and URM:
   a. The adapter can monitor individual batch tasks to determine whether or not each batch submission was processed successfully and obtain a listing of errors if a task fails. For details, see Checking the Status of Individual Batch Tasks (page 6-2).
   b. The adapter can also check the status of multiple batch tasks. However, the adapter must check the status of an individual batch task to obtain a listing of any errors for that task. For details, see Checking the Status of Multiple Batch Tasks (page 6-5).
   c. The adapter can handle errors when trying to communicate with URM. For details, see Handling Task Status Errors (page 6-7).
   d. The adapter can process lists that are too long to transmit in one request by processing the response in chunks. For details, see Segmenting Response Data (page 6-9).
   e. The adapter can upload log files to URM. For details, see Uploading External Log Files (page 6-11).
   f. The adapter can ping the URM server to see if it is running. For details, see Pinging the URM Server (page 6-12).

Audience

This guide is intended for application developers who need to develop URM adapters.

Conventions

The following conventions are used throughout this guide:

❖ Forward slashes (/) are used to separate parts of an Internet address. For example, http://www.oracle.com/en/index.htm. A forward slash might or might not appear at the end of an Internet address.

❖ Backward slashes (\) are used to separate the levels in a path to a Windows server, directory, or file. For example, C:\stellent\idcm1\. A backward slash will always appear after the end of a Windows server, directory, or file path.

❖ Forward slashes (/) are also used to separate the levels in a path to a UNIX server, directory, or file. For example, /usr/stellent/idcm1.

❖ File names and file paths within text are indicated by the following convention: <*filename*> file in the <*path_to_directory*> directory.

❖ The notation <*install_dir*> is used to refer to the location on your system where the URM Server instance is installed.

❖ Notes, technical tips, important notices, and cautions use these conventions:

Symbols	**Description**
💡	This is a note. It is used to bring special attention to information.
⚙	This is a technical tip. It is used to identify information that can be used to make your tasks easier.
❗	This is an important notice. It is used to identify a required step or required information.
✖	This is a caution. It is used to identify information that might cause loss of data or serious system problems.

INTEGRATION METHODS

The URM adapter services documented in this guide can be accessed by any external program or HTML page using a wide variety of protocols. URM can be integrated with other enterprise applications using the following integration methods:

❖ Java API (IdcCommand) integration using the IdcCommand Java Command Utility. For more information, refer to the *Idc Command Reference Guide* provided with Content Server.

❖ Component Object Model (COM) integration using the ActiveX utility or the IntradocClient OCX component. For more information, refer to the *Idc Command Reference Guide* provided with Content Server.

❖ Java Server Page (JSP) integration from a JSP running in Content Server, a JSP through the Content Server JavaBean, or a JSP through the Content Server Enterprise JavaBean (EJB) deployed on your J2EE application server. For more information, refer to the documentation provided with Content Integration Suite (CIS), which is a separate product.

❖ Java 2 Enterprise Edition API (J2EE) integration by deploying the Content Server Enterprise JavaBean on your J2EE-compliant application server. For more information, refer to the documentation provided with CIS.

- Open Document Management API (ODMA) integration using the ODMAbased plug-in. For more information, refer to the documentation provided with Desktop Integration Suite, which is a separate product.
- Simple Object Access Protocol (SOAP) integration using the SOAP protocol. For more information, refer to the *Using WSDL Generator and SOAP* guide. This guide is included in the *WsdlGenerator.zip* file provided with URM.
- Virtual Folders integration using the Folders component. For more information, refer to the *Folders and WebDAV Administration Guide* provided with the Folders/WebDAV component.
- Web Distributed Authoring and Versioning (WebDAV) integration using the WebDAV component. For more information, refer to the *Folders and WebDAV Administration Guide* provided with the Folders/WebDAV component.

Chapter

2

ADAPTER REGISTRATION

OVERVIEW

This section covers the following topics:

Registering Adapters

The adapter must register itself to URM so that URM knows about the repository and is ready to manage content stored within that repository. The adapter registration process is as follows (see Figure 2-1):

- **(AR01)** An administrator creates a user for the adapter in URM. The adapter will use that user's credentials to log onto URM before interacting with URM. In environments where a single sign on solution, such as Obelix, is in use, there might be no need to create a user. However, the administrator would still have to configure the adapter user with the correct role and rights to allow that user to get adapter privileges in URM. In most cases, using the rmaadmin role is sufficient. However, the correct role and rights will depend on the purpose of the adapter.
- **(AR02)** The administrator installs the adapter and gives it the previously-created user credentials. The administrator must provide a source name (dSource) associated with the repository and can provide a table name (dTable) to be used when creating tables for the records or non-record items contained within the repository. If the optional dTable parameter is not specified, dSource will be used for dTable.

  The maximum length for dSource is 30 characters, and the maximum length for dTable is 16 characters. If dTable is not specified, dSource must not exceed 16 characters or dTable will fail its maximum size requirement.

  The administrator also has to configure the adapter so that it knows the repository with which it will be associated. This configuration will vary from adapter to adapter.
- **(AR03)** The newly-installed adapter gathers a listing of the metadata values associated with items in the repository with which it is associated.
- **(AR04)** The adapter prompts the administrator for a Uniform Resource Identifier (URI) to use when connecting to URM.
- **(AR05)** The administrator provides the URI.
- **(AR06)** The adapter connects to URM, authenticates itself, and calls the GET_EXTERNAL_DEFAULT_FIELDS service to request a listing of the default metadata that URM expects from adapters. For additional details, see GET_EXTERNAL_DEFAULT_FIELDS (page A-47).

- ❖ **(AR07)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the default metadata fields to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- ❖ **(AR08)** The adapter checks the status response.
- ❖ **(AR09)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- ❖ **(AR10)** If there is no error, the adapter receives the default metadata fields from URM. The adapter maps the repository's metadata fields to the default metadata fields, creating an externalFieldsMap result set. Please note the following important considerations when creating this result set:
  - The externalFieldsMap result set must map the adapter's unique ID field to either URM's dDocName or dLongName field. By default, dDocName is 100 characters long. If an adapter needs a long ID field, the optional keysize parameter must be specified. If the optional keysize parameter is specified, dLongName must be used.
  - By default, dDocTitle is 200 characters long.
  - If you want to maintain the original field data when it needs to be truncated, consider creating a custom field, with the same name as the mapped field, that is long enough to hold the original data.

    For example, consider you have a Subject field that you map to dDocTitle, which is indicated as a truncated field. You could create a custom field named Subject. Then you only need to send the Subject field value once, as URM will truncate the data if it is greater than the length of the field for dDocTitle. The full value will still be in the Subject field.

  The externalFieldsMap result set will be passed to URM with the CREATE_EXTERNAL_SOURCE service (AR12).

❖ **(AR11)** If necessary, the adapter collates any repository metadata that cannot be mapped to create a custom metadata listing, creating an externalCustomMetaDefinition result set. Please note the following important considerations when creating this result set:

- dOptionListType—only the following values are valid: combo, multi, or strict
- dType—only the following values are valid: Date, Int, Text, BigText, or Memo
- dIsDisplayOnly—indicates if the field is a placeholder field. If the field is defined as a placeholder field, URM does not create the field in the source table.
- dLength—specifies the length of the field. The following are the length ranges and default lengths for custom fields (if the length is not specified, URM will use the default length for each field type):
  - Text field: 1–100; default 100
  - BigText: 101–200; default 200
  - Memo field: 201–2Gb (MS SQL), 201–4000 (Oracle),
    201–2000 (Oracle Japanese); default 1000 for all
- If you are using Microsoft SQL Server 2005, you should set the EnableLongMemoFieldForSCS7 configuration variable in the content server instance where URM is installed.

  When the EnableLongMemoFieldForSCS7 configuration variable is set to true, during table creation the length of the Memo type is 2,000,000,000. When this configuration variable is set to false, the length of the Memo type is 1000 (note that is specific to URM; the default Memo type length for Content Server is 255).

  It is recommended that you set Content Server configuration variables using Admin Server. Content Server configuration variables can also be set by editing the *config.cfg* file located in the *<install_dir>/config* directory.

Note: When using any database other than Microsoft SQL Server 2005, the EnableLongMemoFieldForSCS7 configuration variable has no effect on content server functionality.

The externalCustomMetaDefinition result set will be passed to URM with the CREATE_EXTERNAL_SOURCE service (AR12).

❖ **(AR12)** The adapter calls the CREATE_EXTERNAL_SOURCE service to request that URM create an external source.

  The adapter must provide the following parameters:

  - dSource
  - externalFieldMap (result set)

  The adapter can specify the following parameters

  - dTable
  - externalCustomMetaDefinition (result set)
  - blocksize
  - keysize
  - sourceDisplayName

  For additional details, see CREATE_EXTERNAL_SOURCE (page A-29).

❖ **(AR13)** URM creates the external source.

❖ **(AR14)** URM sends a status response. The status response either indicates that the creation was successful (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).

❖ **(AR15)** The adapter receives and evaluates the status response.

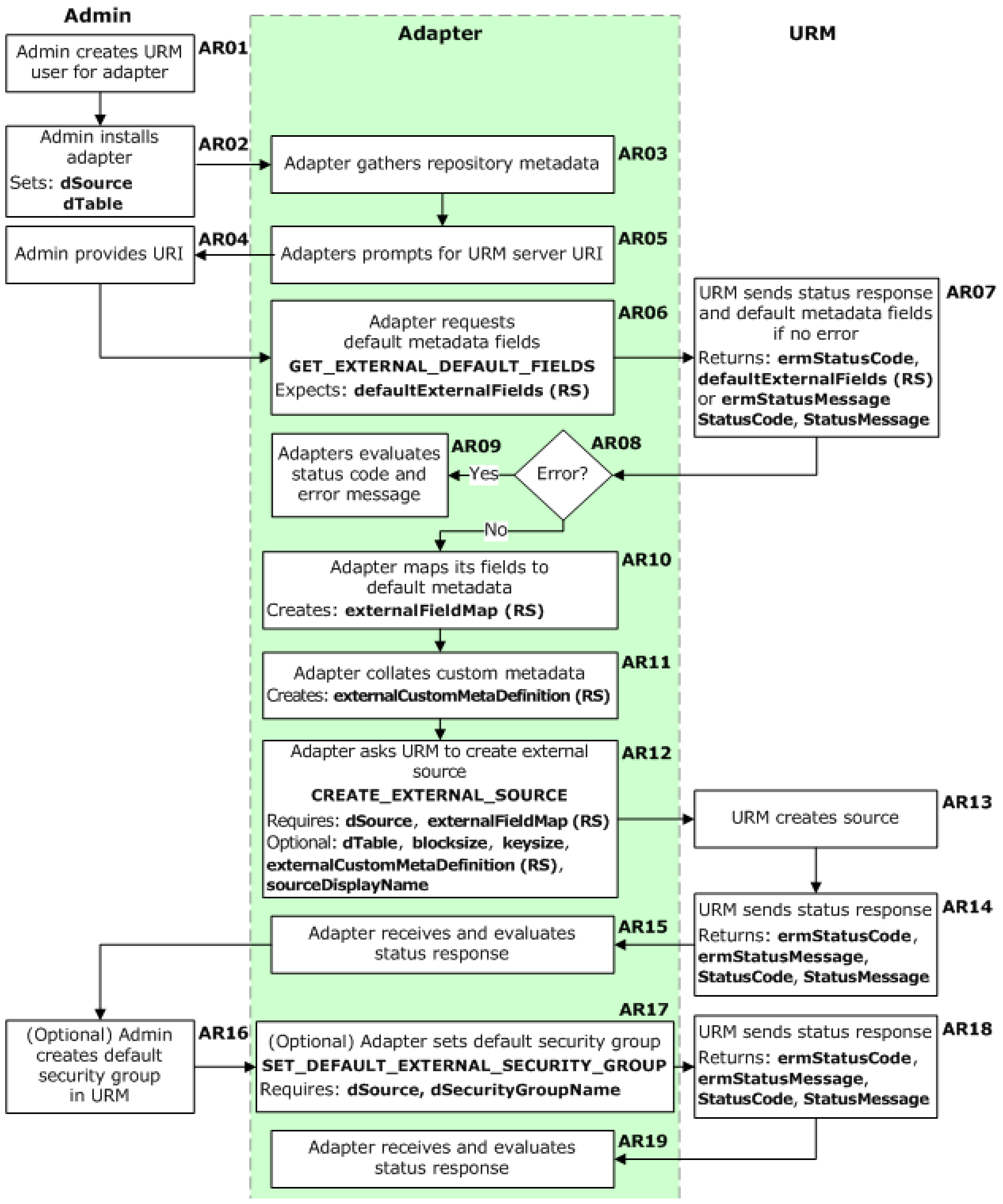
❖ **(AR16)** Optionally, the administrator can create a default security group in URM.

❖ **(AR17)** The adapter can call the SET_DEFAULT_EXTERNAL_SECURITY_GROUP service to set the default security group. If a default security group is not set, the Public security group is used if a security group is not specified when items are declared (external checkin). For additional details, see SET_DEFAULT_EXTERNAL_SECURITY_GROUP (page A-80).

❖ **(AR18)** URM sends a status response. The status response either indicates that the setting of the default security group was successful (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).

❖ **(AR19)** The adapter receives and evaluates the status response.

Figure 2-1 Adapter registration


Admin
Adapter
URM
Admin creates URM user for adapter
AR01
Admin installs adapter
Sets: dSource dTable
AR02
Adapter gathers repository metadata
AR03
Admin provides URI
AR04
Adapters prompts for URM server URI
AR05
Adapter requests default metadata fields
GET_EXTERNAL_DEFAULT_FIELDS
Expects: defaultExternalFields (RS)
AR06
URM sends status response and default metadata fields if no error
Returns: ermStatusCode, defaultExternalFields (RS) or ermStatusMessage StatusCode, StatusMessage
AR07
Error?
AR08
Yes
Adapters evaluates status code and error message
AR09
No
Adapter maps its fields to default metadata
Creates: externalFieldMap (RS)
AR10
Adapter collates custom metadata
Creates: externalCustomMetaDefinition (RS)
AR11
Adapter asks URM to create external source
CREATE_EXTERNAL_SOURCE
Requires: dSource, externalFieldMap (RS)
Optional: dTable, blocksize, keysize, externalCustomMetaDefinition (RS), sourceDisplayName
AR12
URM creates source
AR13
URM sends status response
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
AR14
Adapter receives and evaluates status response
AR15
(Optional) Admin creates default security group in URM
AR16
(Optional) Adapter sets default security group
SET_DEFAULT_EXTERNAL_SECURITY_GROUP
Requires: dSource, dSecurityGroupName
AR17
URM sends status response
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
AR18
Adapter receives and evaluates status response
AR19

UPDATING EXTERNAL FIELD MAPPINGS

When an adapter is registered, the adapter must provide an externalFieldsMap result set that maps the repository's metadata fields to the default URM metadata fields. At the least, this result set must map the adapter's unique ID field to either URM's dDocName or dLongName field. For details, see Registering Adapters (page 2-2).

To update external field mappings after registering the adapter, complete the following steps:

- ❖ **(UM01)** If necessary, the adapter can call the GET_EXTERNAL_DEFAULT_FIELDS service to request a listing of the default metadata that URM expects from adapters. For additional details, see GET_EXTERNAL_DEFAULT_FIELDS (page A-47).
- ❖ **(UM02)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the default metadata fields to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- ❖ **(UM03)** The adapter checks the status response.
- ❖ **(UM04)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- ❖ **(UM05)** If there is no error, if necessary the adapter can also call the GET_EXTERNAL_TABLE_FIELDS service to request a listing of existing metadata definitions. For additional details, see GET_EXTERNAL_TABLE_FIELDS (page A-51).
- ❖ **(UM06)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the metadata definitions to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- ❖ **(UM07)** The adapter checks the status response.
- ❖ **(UM08)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- ❖ **(UM09)** If there is no error, the adapter maps the repository's metadata fields to the default metadata fields, creating an externalFieldsMap result set. Please note the following important considerations when creating this result set:
  - The externalFieldsMap result set contains the columns rmField, externalField, and externalCaption; specifying the fields mapping between URM fields and the adapter's fields.

- If rmField already exists, the mapped externalField and externalCaption are updated. Making changes to external field mappings never results in database changes; it only affects the metadata mapping for future items. The metadata mapping for any existing items will not be changed by updating the external field mappings.
- If you want to maintain the original field data when it needs to be truncated, consider creating a custom field, with the same name as the mapped field, that is long enough to hold the original data.

  For example, consider you have a Subject field that you map to dDocTitle, which is indicated as a truncated field. You could create a custom field named Subject. Then you only need to send the Subject field value once, as URM will truncate the data if it is greater than the length of the field for dDocTitle. The full value will still be in the Subject field.

The externalFieldsMap result set will be passed to URM with the UPDATE_EXTERNAL_FIELD_MAPPING service (UM10).

Important: Making changes to external field mappings never results in database changes; it only affects the metadata mapping for future items.

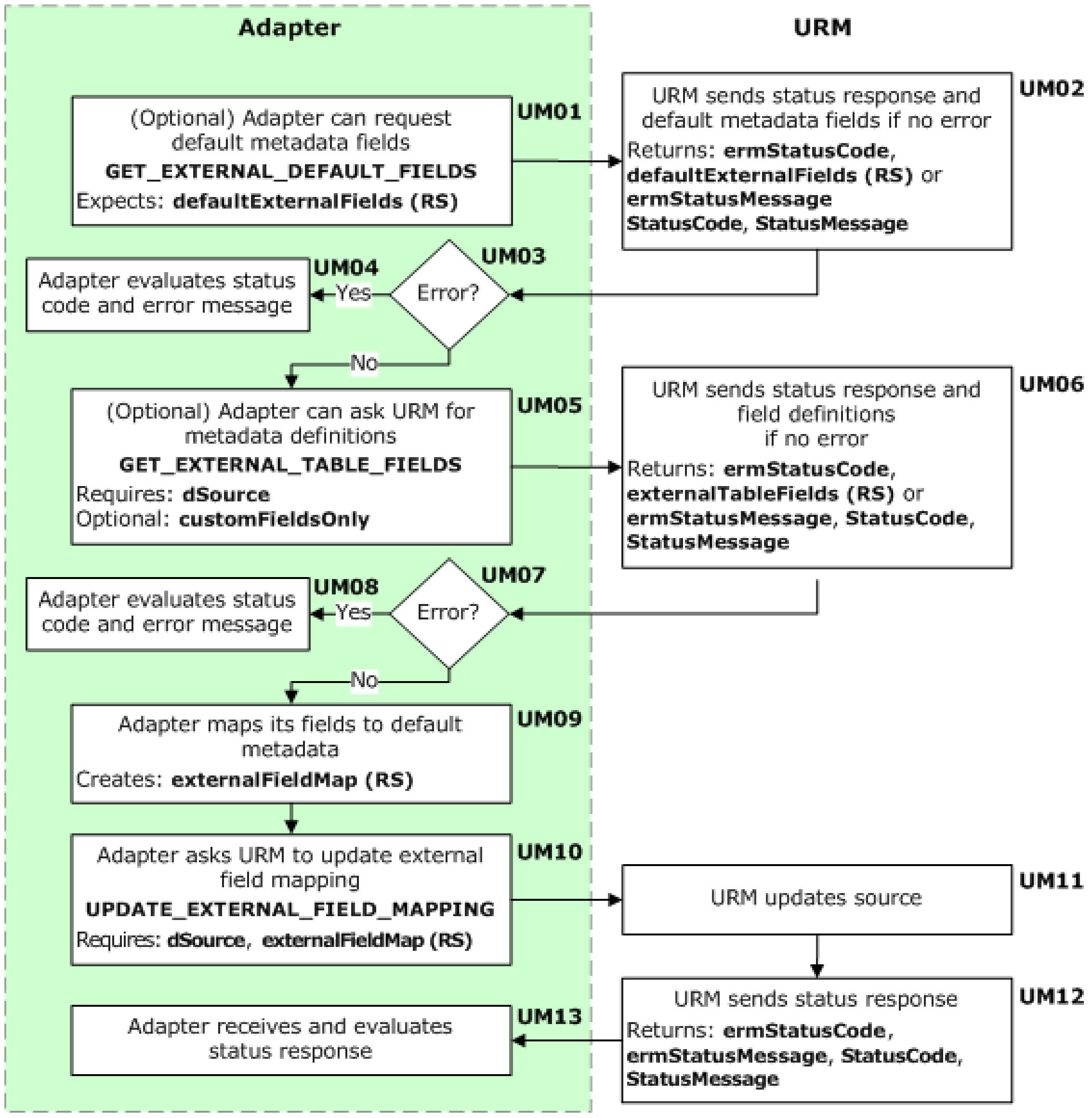
❖ **(UM10)** The adapter calls the UPDATE_EXTERNAL_FIELD_MAPPING service to request that URM update the source. For additional details, see UPDATE_EXTERNAL_FIELD_MAPPING (page A-89).

❖ **(UM11)** URM updates the source.

❖ **(UM12)** URM sends a status response. The status response either indicates that the update was successful (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).

❖ **(UM13)** The adapter receives and evaluates the status response.

Figure 2-2 Updating the external field mapping


Adapter
URM
(Optional) Adapter can request default metadata fields
GET_EXTERNAL_DEFAULT_FIELDS
Expects: defaultExternalFields (RS)
UM01
URM sends status response and default metadata fields if no error
Returns: ermStatusCode, defaultExternalFields (RS) or ermStatusMessage StatusCode, StatusMessage
UM02
UM03
Error?
UM04
Yes
Adapter evaluates status code and error message
No
(Optional) Adapter can ask URM for metadata definitions
GET_EXTERNAL_TABLE_FIELDS
Requires: dSource
Optional: customFieldsOnly
UM05
URM sends status response and field definitions if no error
Returns: ermStatusCode, externalTableFields (RS) or ermStatusMessage, StatusCode, StatusMessage
UM06
UM07
Error?
UM08
Yes
Adapter evaluates status code and error message
No
Adapter maps its fields to default metadata
Creates: externalFieldMap (RS)
UM09
Adapter asks URM to update external field mapping
UPDATE_EXTERNAL_FIELD_MAPPING
Requires: dSource, externalFieldMap (RS)
UM10
URM updates source
UM11
URM sends status response
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
UM12
Adapter receives and evaluates status response
UM13

ADDING EXTERNAL CUSTOM FIELDS

When an adapter is registered, the adapter can collate any repository metadata that cannot be mapped to default URM metadata fields, creating an externalCustomMetaDefinition result set (creating a custom metadata listing). For details, see Registering Adapters (page 2-2).

To add external custom fields after registering the adapter, complete the following steps:

- ❖ **(AC01)** If necessary, the adapter can call the GET_EXTERNAL_DEFAULT_FIELDS service to request a listing of the default metadata that URM expects from adapters. For additional details, see GET_EXTERNAL_DEFAULT_FIELDS (page A-47).
- ❖ **(AC02)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the default metadata fields to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- ❖ **(AC03)** The adapter checks the status response.
- ❖ **(AC04)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- ❖ **(AC05)** If there is no error, if necessary the adapter can also call the GET_EXTERNAL_TABLE_FIELDS service to request a listing of existing metadata definitions. For additional details, see GET_EXTERNAL_TABLE_FIELDS (page A-51).
- ❖ **(AC06)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the metadata definitions to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- ❖ **(AC07)** The adapter checks the status response.
- ❖ **(AC08)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- ❖ **(AC09)** If there is no error, if necessary the adapter can also call the GET_MAXIMUM_FIELD_LENGTHS service to determine the maximum field length for each text type. For additional details, see GET_MAXIMUM_FIELD_LENGTHS (page A-59).
- ❖ **(AC10)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the maximum field lengths to the adapter. If there is an error, URM sends an error message (ermStatusMessage).

  The maximumFieldLengths result set lists the maximum field length for each text type. This result set contains two columns:

  ```
  type
  length
  ```

The following text types are returned: Text, BigText, and Memo. The length value for each text type represents the maximum length for any field of that type.

- ❖ **(AC11)** The adapter checks the status response.
- ❖ **(AC12)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- ❖ **(AC13)** If there is no error, the adapter collates a list of additional custom metadata fields, creating an externalCustomMetaDefinition result set. Please note the following important considerations when creating this result set:
  - If a field already exists, you will receive an error.
  - dOptionListType—only the following values are valid: combo, multi, or strict
  - dType—only the following values are valid: Date, Int, Text, BigText, or Memo
  - dIsDisplayOnly—indicates if the field is a placeholder field. If the field is defined as a placeholder field, URM does not create the field in the source table.
  - If you want to maintain the original field data when it needs to be truncated, consider creating a custom field, with the same name as the mapped field, that is long enough to hold the original data.

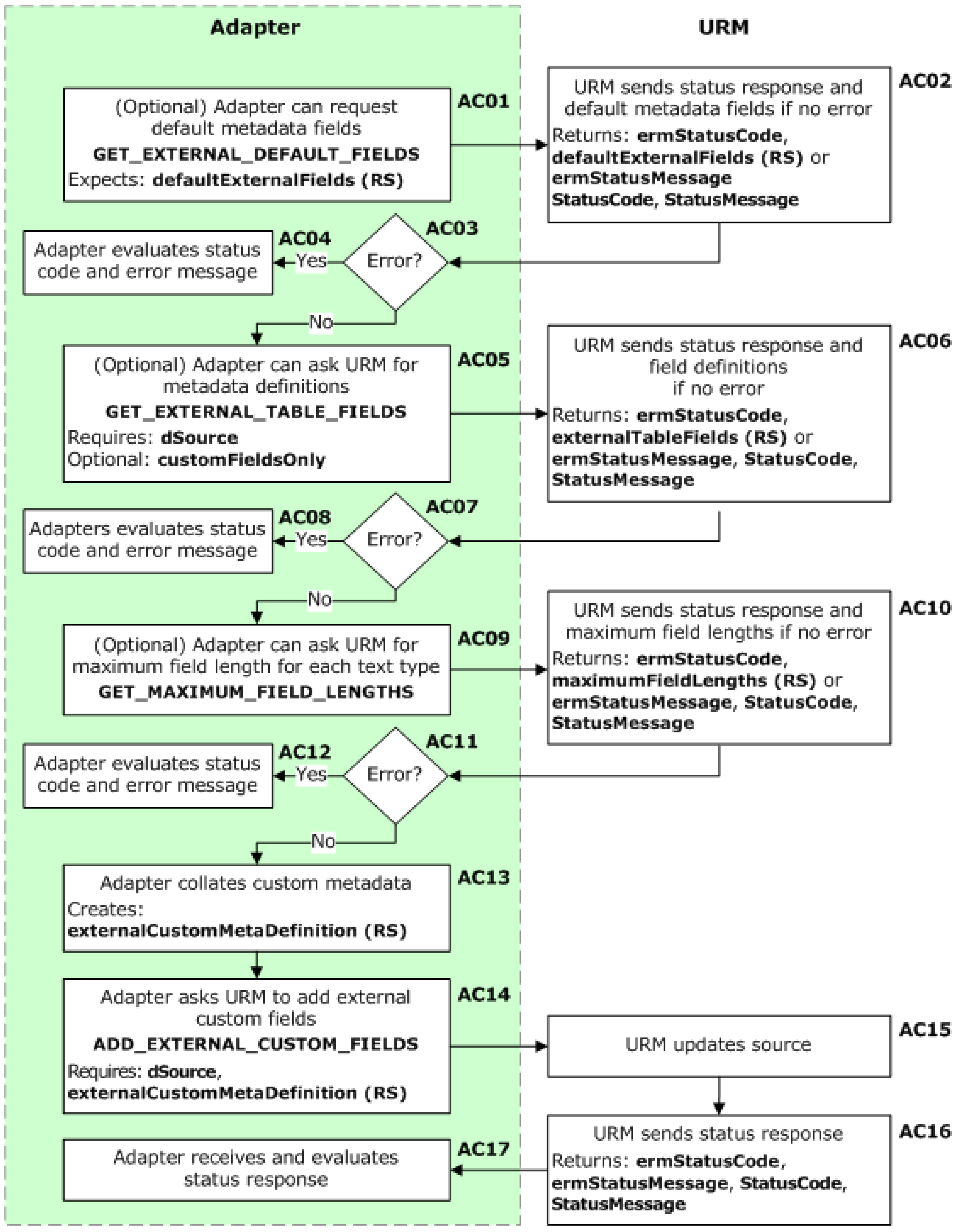
    For example, consider you have a Subject field that you map to dDocTitle, which is indicated as a truncated field. You could create a custom field named Subject. Then you only need to send the Subject field value once, as URM will truncate the data if it is greater than the length of the field for dDocTitle. The full value will still be in the Subject field.

  The externalCustomMetaDefinition result set will be passed to URM with the ADD_EXTERNAL_CUSTOM_FIELDS service (AC10).

Important: Making changes to external custom fields never results in database changes; it only affects the metadata mapping for future items.

- ❖ **(AC14)** The adapter calls the ADD_EXTERNAL_CUSTOM_FIELDS service to request that URM update the source. For additional details, see ADD_EXTERNAL_CUSTOM_FIELDS (page A-5).
- ❖ **(AC15)** URM updates the source.
- ❖ **(AC16)** URM sends a status response. The status response either indicates that the update was successful (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).
- ❖ **(AC17)** The adapter receives and evaluates the status response.

Figure 2-3 Adding external custom fields


Adapter
URM
(Optional) Adapter can request default metadata fields
GET_EXTERNAL_DEFAULT_FIELDS
Expects: defaultExternalFields (RS)
AC01
URM sends status response and default metadata fields if no error
Returns: ermStatusCode, defaultExternalFields (RS) or ermStatusMessage StatusCode, StatusMessage
AC02
Error?
AC03
Adapter evaluates status code and error message
AC04
Yes
No
(Optional) Adapter can ask URM for metadata definitions
GET_EXTERNAL_TABLE_FIELDS
Requires: dSource
Optional: customFieldsOnly
AC05
URM sends status response and field definitions if no error
Returns: ermStatusCode, externalTableFields (RS) or ermStatusMessage, StatusCode, StatusMessage
AC06
Error?
AC07
Adapters evaluates status code and error message
AC08
Yes
No
(Optional) Adapter can ask URM for maximum field length for each text type
GET_MAXIMUM_FIELD_LENGTHS
AC09
URM sends status response and maximum field lengths if no error
Returns: ermStatusCode, maximumFieldLengths (RS) or ermStatusMessage, StatusCode, StatusMessage
AC10
Error?
AC11
Adapter evaluates status code and error message
AC12
Yes
No
Adapter collates custom metadata
Creates: externalCustomMetaDefinition (RS)
AC13
Adapter asks URM to add external custom fields
ADD_EXTERNAL_CUSTOM_FIELDS
Requires: dSource, externalCustomMetaDefinition (RS)
AC14
URM updates source
AC15
URM sends status response
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
AC16
Adapter receives and evaluates status response
AC17

UPDATING EXTERNAL CUSTOM FIELDS

When an adapter is registered, the adapter can collate any repository metadata that cannot be mapped to default URM metadata fields, creating an externalCustomMetaDefinition result set (creating a custom metadata listing). For details, see Registering Adapters (page 2-2).

To update external custom fields after registering the adapter, complete the following steps:

❖ **(UC01)** If necessary, the adapter can call the GET_EXTERNAL_DEFAULT_FIELDS service to request a listing of the default metadata that URM expects from adapters. For additional details, see GET_EXTERNAL_DEFAULT_FIELDS (page A-47).

❖ **(UC02)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the default metadata fields to the adapter. If there is an error, URM sends an error message (ermStatusMessage).

❖ **(UC03)** The adapter checks the status response.

❖ **(UC04)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.

❖ **(UC05)** If there is no error, if necessary the adapter can also call the GET_EXTERNAL_TABLE_FIELDS service to request a listing of existing metadata definitions. For additional details, see GET_EXTERNAL_TABLE_FIELDS (page A-51).

❖ **(UC06)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the metadata definitions to the adapter. If there is an error, URM sends an error message (ermStatusMessage).

❖ **(UC07)** The adapter checks the status response.

❖ **(UC08)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.

❖ **(UC09)**If there is no error, the adapter collates an updated list of existing custom metadata fields, creating an externalCustomMetaDefinition result set. Please note the following important considerations when creating this result set:

- If a field does not exist already, you will receive an error.
- dOptionListType—only the following values are valid: combo, multi, or strict
- dType—only the following values are valid: Date, Int, Text, BigText, or Memo

- dIsDisplayOnly—indicates if the field is a placeholder field. If the field is defined as a placeholder field, URM does not create the field in the source table.

The externalCustomMetaDefinition result set will be passed to URM with the UPDATE_EXTERNAL_CUSTOM_FIELDS service (AC10).

Important: Making changes to external custom fields never results in database changes; it only affects the metadata mapping for future items.

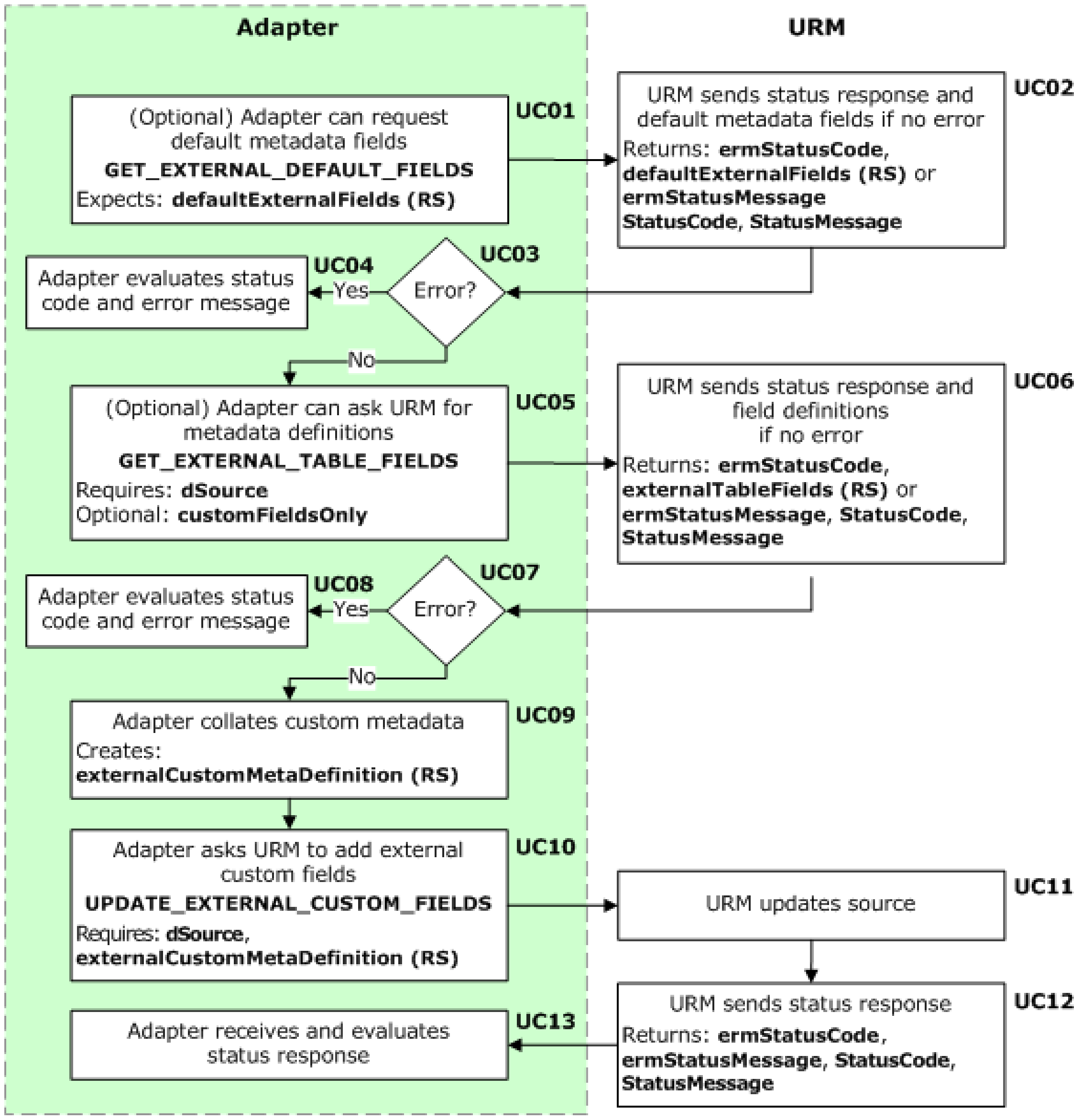
❖ **(UC10)** The adapter calls the UPDATE_EXTERNAL_CUSTOM_FIELDS service to request that URM update the source. For additional details, see UPDATE_EXTERNAL_CUSTOM_FIELDS (page A-85).

❖ **(UC11)** URM updates the source.

❖ **(UC12)** URM sends a status response. The status response either indicates that the update was successful (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).

❖ **(UC13)** The adapter receives and evaluates the status response.

Figure 2-4 Updating external custom fields


Adapter
URM
(Optional) Adapter can request default metadata fields
GET_EXTERNAL_DEFAULT_FIELDS
Expects: defaultExternalFields (RS)
UC01
URM sends status response and default metadata fields if no error
Returns: ermStatusCode, defaultExternalFields (RS) or ermStatusMessage StatusCode, StatusMessage
UC02
Error?
UC03
Adapter evaluates status code and error message
UC04
Yes
No
(Optional) Adapter can ask URM for metadata definitions
GET_EXTERNAL_TABLE_FIELDS
Requires: dSource
Optional: customFieldsOnly
UC05
URM sends status response and field definitions if no error
Returns: ermStatusCode, externalTableFields (RS) or ermStatusMessage, StatusCode, StatusMessage
UC06
Error?
UC07
Adapter evaluates status code and error message
UC08
Yes
No
Adapter collates custom metadata
Creates: externalCustomMetaDefinition (RS)
UC09
Adapter asks URM to add external custom fields
UPDATE_EXTERNAL_CUSTOM_FIELDS
Requires: dSource, externalCustomMetaDefinition (RS)
UC10
URM updates source
UC11
URM sends status response
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
UC12
Adapter receives and evaluates status response
UC13

SETTING UP THE EXTRA METADATA FIELD

To activate the extra metadata field after registering the adapter, complete the following steps:

- **(SE01)** The adapter calls the SETUP_EXTRA_METADATA_FIELD service to request that URM activate the extra metadata field. URM creates an extra table with a dExtraMetaData column. It is then possible to pass a dExtraMetaData value during future item checkins or edits.

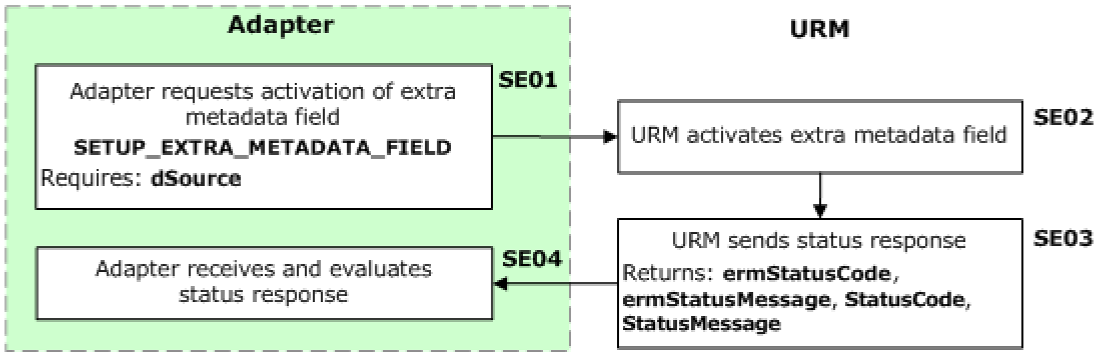
  It is currently recommended that you pass an XML string with extra metadata field mappings for dExtraMetaData. The dExtraMetaData string would be included along with all other field mapping data; this is just a special field that is not created or mapped.

  For additional details, see SETUP_EXTRA_METADATA_FIELD (page A-81).

Important: Once SETUP_EXTRA_METADATA_FIELD is called and the extra metadata field is created, it cannot be removed.

- **(SE02)** URM activates the extra metadata field.
- **(SE03)** URM sends a status response. The status response either indicates that the activation was successful (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).
- **(SE04)** The adapter receives and evaluates the status response.

Figure 2-5 Setting up the extra metadata field


Adapter
URM
Adapter requests activation of extra metadata field
SETUP_EXTRA_METADATA_FIELD
Requires: dSource
SE01
URM activates extra metadata field
SE02
URM sends status response
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
SE03
Adapter receives and evaluates status response
SE04

Chapter
3

Repository Monitoring

Overview

This section covers the following topics:

DECLARING ITEMS TO URM (EXTERNAL CHECKIN)

As new items are added to the repository, the adapter should notify URM of their existence and provide URM with enough metadata to allow URM to manage the retention of the items. Since URM has some of the capability of Content Server, the administrator can define a profile that derives additional metadata within URM after the adapter has provided its metadata. The records can be preserved in place if the repository has the ability to ensure that the record will remain unalterable during the retention period. The repository also needs to be able to purge the records at the end of the retention period. Adapters associated with repositories that cannot manage records as described have to move the records to URM for proper preservation and disposition. For details, see Checking Items into URM (Internal Checkin) (page 3-22).

This section covers the following topics:

- Declaring Items in Batch (page 3-2)
- Declaring Individual Items (page 3-5)

Declaring Items in Batch

New items can be declared to URM in a batch (external checkin; the items are stored externally in the repository, not in URM). The batch declaration process is as follows (see Figure 3-1):

- **(AB01)** The adapter gathers a listing and the metadata of items to be declared.
- **(AB02)** The adapter calls the GET_EXTERNAL_TABLE_FIELDS service to ask URM for metadata definitions so that the adapter knows how to format the metadata it is about to send to URM. For additional details, see GET_EXTERNAL_TABLE_FIELDS (page A-51).
- **(AB03)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the metadata definitions to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- **(AB04)** The adapter checks the status response.
- **(AB05)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(AB06)** If there is no error, the adapter receives the metadata definitions from URM. The adapter uses the metadata definitions and the listing and metadata it has collected to build a declaration request.

- ❖ **(AB07)** The adapter calls the CHECKIN_MULTIPLE_EXTERNAL service to declare the batch of items to URM (external checkin). Please note the following important consideration:

  - A record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error.

  For additional details, see CHECKIN_MULTIPLE_EXTERNAL (page A-21).

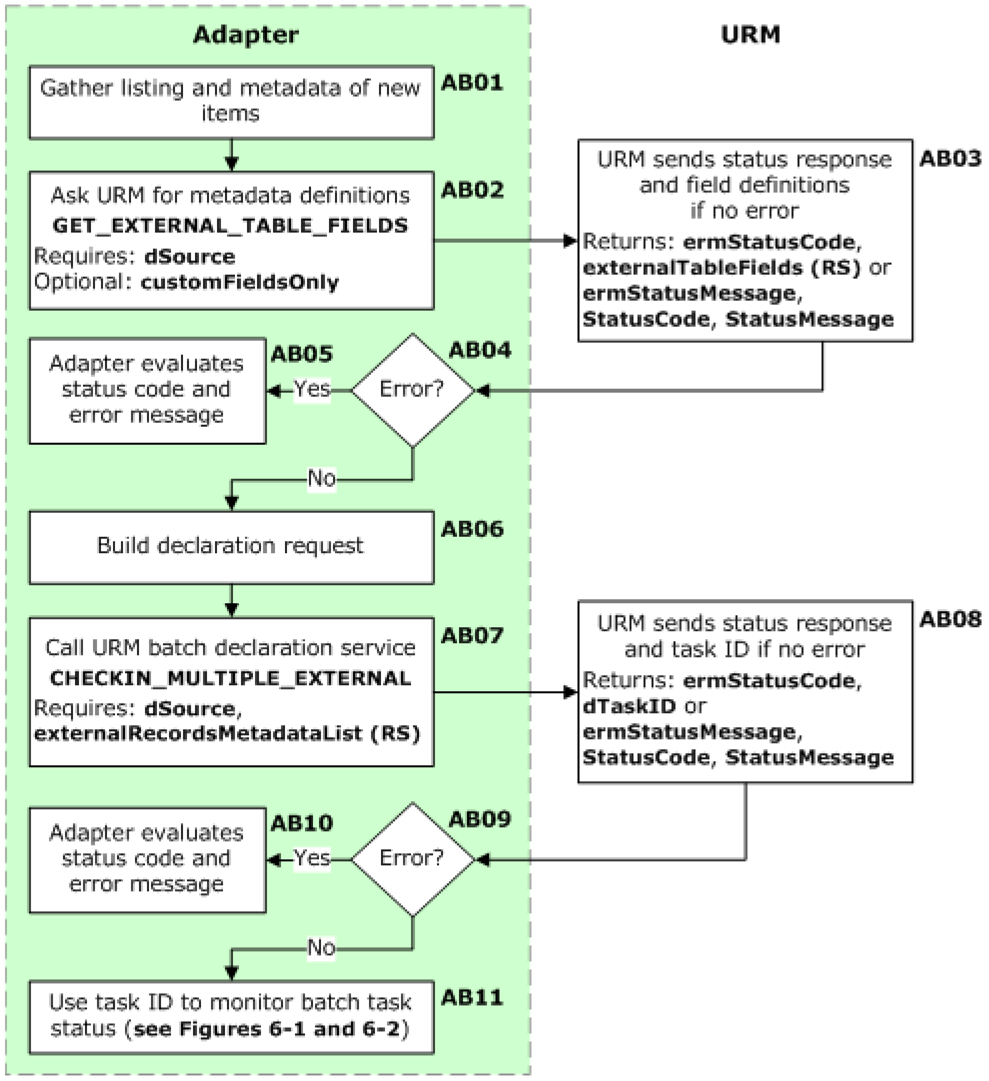
- ❖ **(AB08)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the adapter a task ID. If there is an error, URM sends an error message (ermStatusMessage).

- ❖ **(AB09)** The adapter checks the status response.

- ❖ **(AB10)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.

- ❖ **(AB11)** If there is no error, the adapter receives a task ID from URM. The adapter uses the task ID to monitor the status of the batch checkin. For details, see Checking the Status of Individual Batch Tasks (page 6-2) and Checking the Status of Multiple Batch Tasks (page 6-5).

Figure 3-1 Declaring items in batch


Adapter
URM
Gather listing and metadata of new items
AB01
Ask URM for metadata definitions
GET_EXTERNAL_TABLE_FIELDS
Requires: dSource
Optional: customFieldsOnly
AB02
URM sends status response and field definitions if no error
Returns: ermStatusCode, externalTableFields (RS) or ermStatusMessage, StatusCode, StatusMessage
AB03
Error?
AB04
Yes
No
Adapter evaluates status code and error message
AB05
Build declaration request
AB06
Call URM batch declaration service
CHECKIN_MULTIPLE_EXTERNAL
Requires: dSource, externalRecordsMetadataList (RS)
AB07
URM sends status response and task ID if no error
Returns: ermStatusCode, dTaskID or ermStatusMessage, StatusCode, StatusMessage
AB08
Error?
AB09
Yes
No
Adapter evaluates status code and error message
AB10
Use task ID to monitor batch task status (see Figures 6-1 and 6-2)
AB11

Declaring Individual Items

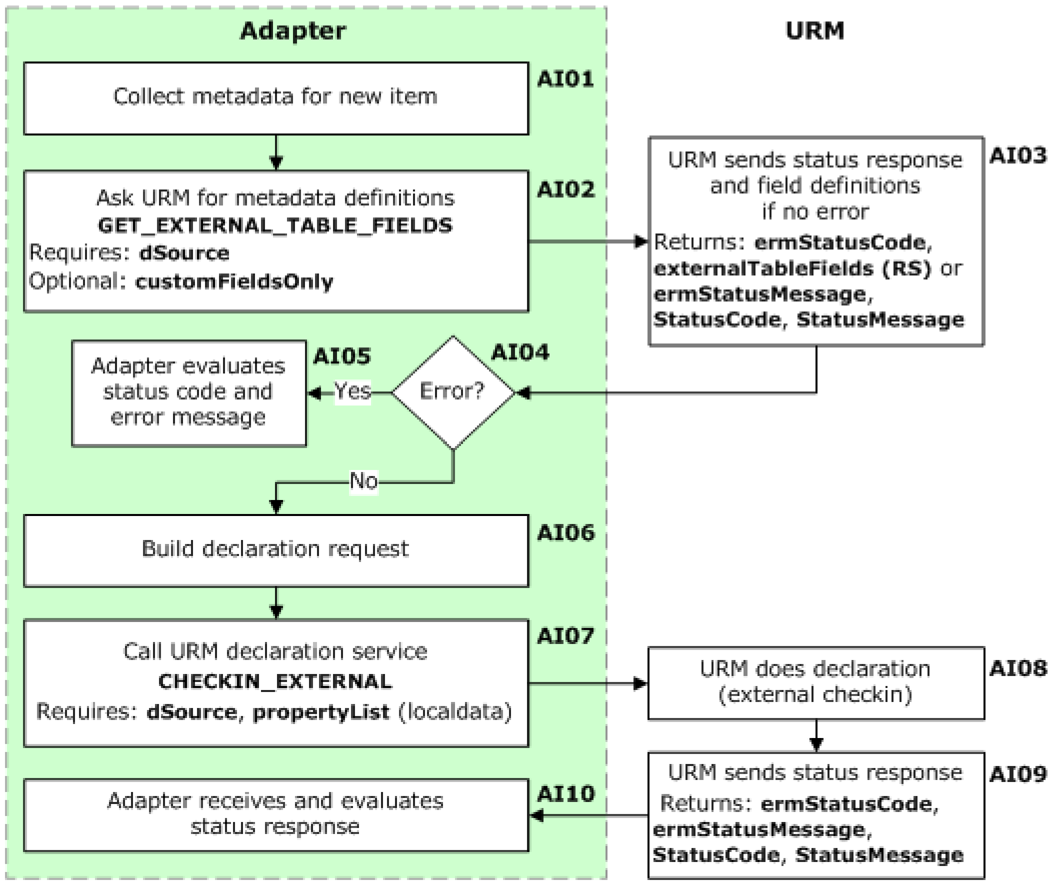
New items can be declared to URM individually (external checkin; the item is stored externally in the repository, not in URM). The individual declaration process is as follows (see Figure 3-2):

- ❖ **(AI01)** The adapter collects metadata for the item to be declared.
- ❖ **(AI02)** The adapter calls the GET_EXTERNAL_TABLE_FIELDS service to ask URM for metadata definitions so that the adapter knows how to format the metadata it is about to send to URM. For additional details, see GET_EXTERNAL_TABLE_FIELDS (page A-51).
- ❖ **(AI03)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the metadata definitions to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- ❖ **(AI04)** The adapter checks the status response.
- ❖ **(AI05)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- ❖ **(AI06)** If there is no error, the adapter receives the metadata definitions from URM. The adapter uses the metadata definitions and the metadata it has collected to build a declaration request.
- ❖ **(AI07)** The adapter calls the CHECKIN_EXTERNAL service to declare the item to URM (external checkin). Please note the following important consideration:

    - A record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error.

    For additional details, see CHECKIN_EXTERNAL (page A-17).
- ❖ **(AI08)** URM does a declaration (external checkin).
- ❖ **(AI09)** URM sends a status response. The status response either indicates that the checkin was successful (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).
- ❖ **(AI10)** The adapter receives and evaluates the status response.

Figure 3-2 Declaring an individual item


Adapter
URM
AI01 Collect metadata for new item
AI02 Ask URM for metadata definitions GET_EXTERNAL_TABLE_FIELDS Requires: dSource Optional: customFieldsOnly
AI03 URM sends status response and field definitions if no error Returns: ermStatusCode, externalTableFields (RS) or ermStatusMessage, StatusCode, StatusMessage
AI04 Error?
Yes
No
AI05 Adapter evaluates status code and error message
AI06 Build declaration request
AI07 Call URM declaration service CHECKIN_EXTERNAL Requires: dSource, propertyList (localdata)
AI08 URM does declaration (external checkin)
AI09 URM sends status response Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
AI10 Adapter receives and evaluates status response

UPDATING ITEMS IN URM

Items within the repository (that have been declared to URM; external checkin) might have their metadata changed, either by users interacting with that repository or by the repository itself. Because that metadata might be used to control retention, it is critical that the adapter send notification of changes to the metadata to URM.

This section covers the following topics:

- ❖ Updating Items in Batch (page 3-7)
- ❖ Updating Individual Items (page 3-10)

Updating Items in Batch

The metadata for items can be updated in URM in a batch (items that have been declared to URM; external checkin). The batch update process is as follows (see Figure 3-3):

- ❖ **(UB01)** The adapter gathers a listing and the metadata of items to be updated.
- ❖ **(UB02)** The adapter calls the GET_EXTERNAL_TABLE_FIELDS service to ask URM for metadata definitions so that the adapter knows how to format the metadata it is about to send to URM. For additional details, see GET_EXTERNAL_TABLE_FIELDS (page A-51).
- ❖ **(UB03)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the metadata definitions to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- ❖ **(UB04)** The adapter checks the status response.
- ❖ **(UB05)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- ❖ **(UB06)** If there is no error, the adapter receives the metadata definitions from URM. The adapter uses the metadata definitions and the listing and metadata it has collected to build an update request. All mapped and custom fields must be included in the update request. Any fields that are left out will be overwritten with an empty string; they will not retain their original value.
- ❖ **(UB07)** The adapter calls the EDIT_MULTIPLE_EXTERNAL service to update the batch of items in URM. Please note the following important considerations:
  - A record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a

value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error.

- Neither dDocName nor dLongName can be updated.

For additional details, see EDIT_MULTIPLE_EXTERNAL (page A-45).

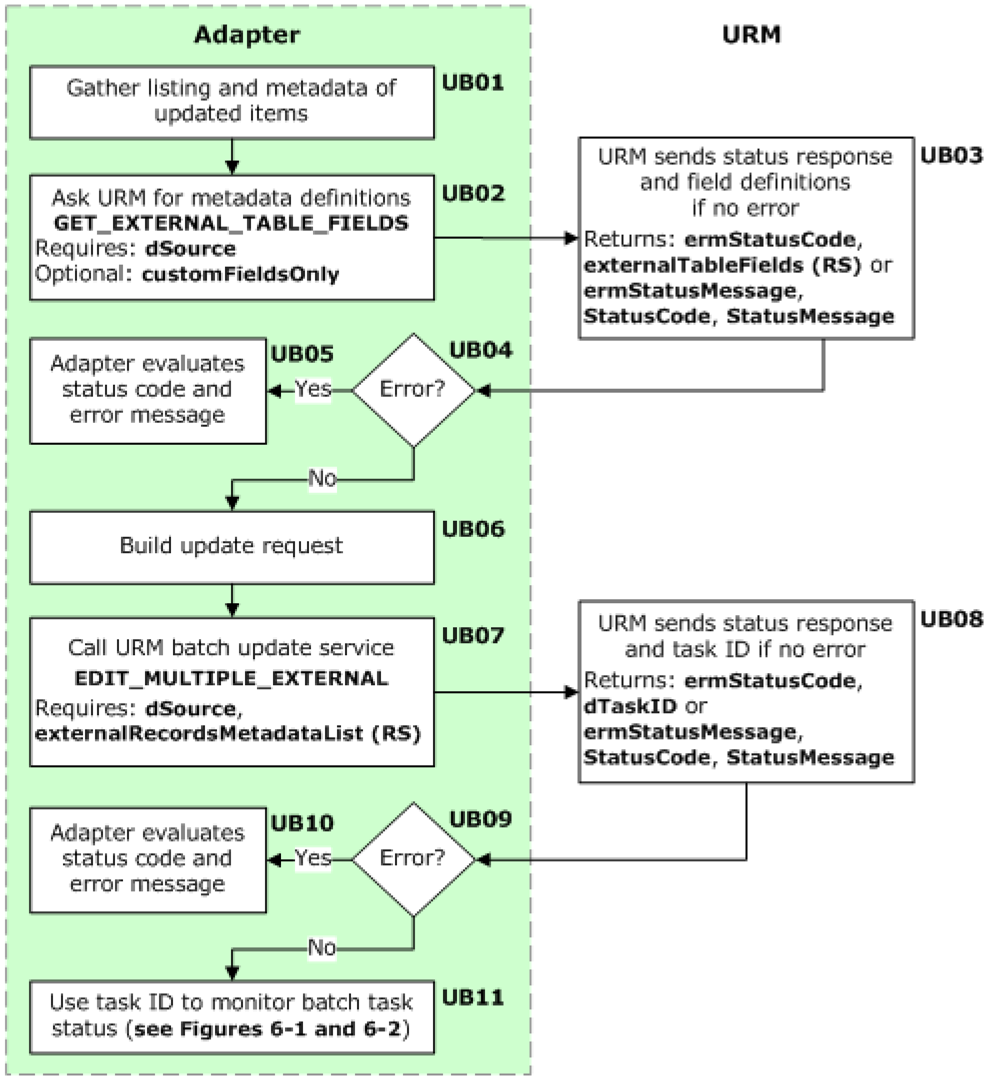
❖ **(UB08)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the adapter a task ID. If there is an error, URM sends an error message (ermStatusMessage).

❖ **(UB09)** The adapter checks the status response.

❖ **(UB10)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.

❖ **(UB11)** If there is no error, the adapter receives a task ID from URM. The adapter uses the task ID to monitor the status of the batch update. For details, see Checking the Status of Individual Batch Tasks (page 6-2) and Checking the Status of Multiple Batch Tasks (page 6-5).

Figure 3-3 Updating items in batch


Adapter
URM
Gather listing and metadata of updated items
UB01
Ask URM for metadata definitions GET_EXTERNAL_TABLE_FIELDS Requires: dSource Optional: customFieldsOnly
UB02
URM sends status response and field definitions if no error Returns: ermStatusCode, externalTableFields (RS) or ermStatusMessage, StatusCode, StatusMessage
UB03
Error?
UB04
Yes
Adapter evaluates status code and error message
UB05
No
Build update request
UB06
Call URM batch update service EDIT_MULTIPLE_EXTERNAL Requires: dSource, externalRecordsMetadataList (RS)
UB07
URM sends status response and task ID if no error Returns: ermStatusCode, dTaskID or ermStatusMessage, StatusCode, StatusMessage
UB08
Error?
UB09
Yes
Adapter evaluates status code and error message
UB10
No
Use task ID to monitor batch task status (see Figures 6-1 and 6-2)
UB11

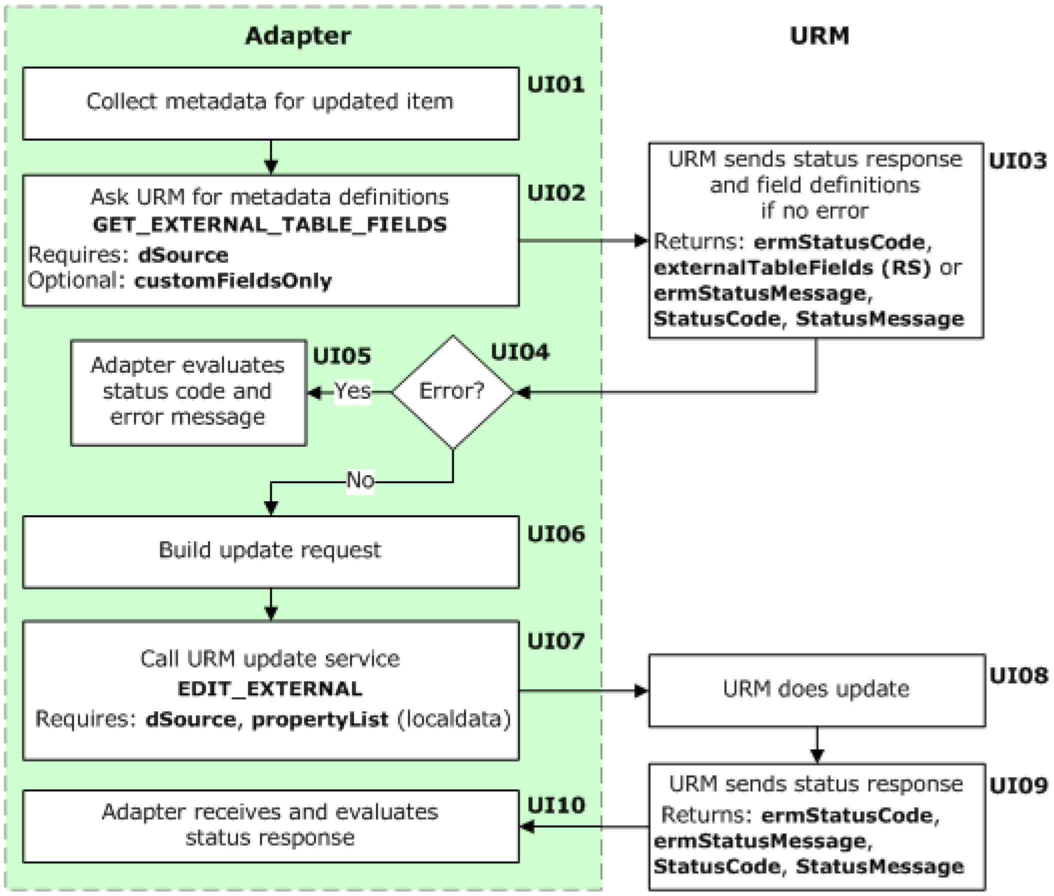
Updating Individual Items

The metadata for a single item can be updated in URM (an item that has been declared to URM; external checkin). The individual update process is as follows (see Figure 3-4):

- ❖ **(UI01)** The adapter collects metadata for the item to be updated.
- ❖ **(UI02)** The adapter calls the GET_EXTERNAL_TABLE_FIELDS service to ask URM for metadata definitions so that the adapter knows how to format the metadata it is about to send to URM. For additional details, see GET_EXTERNAL_TABLE_FIELDS (page A-51).
- ❖ **(UI03)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the metadata definitions to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- ❖ **(UI04)** The adapter checks the status response.
- ❖ **(UI05)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- ❖ **(UI06)** If there is no error, the adapter receives the metadata definitions from URM. The adapter uses the metadata definitions and the metadata it has collected to build an update request. All mapped and custom fields must be included in the update request. Any fields that are left out will be overwritten with an empty string; they will not retain their original value.
- ❖ **(UI07)** The adapter calls the EDIT_EXTERNAL service to update the item in URM. Please note the following important considerations:
  - A record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error.
  - Neither dDocName nor dLongName can be updated.

  For additional details, see EDIT_EXTERNAL (page A-43).
- ❖ **(UI08)** URM does a update.
- ❖ **(UI09)** URM sends a status response. The status response either indicates that the update was successful (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).
- ❖ **(UI10)** The adapter receives and evaluates the status response.

Figure 3-4 Updating individual items


Adapter
URM
Collect metadata for updated item
UI01
Ask URM for metadata definitions
GET_EXTERNAL_TABLE_FIELDS
Requires: dSource
Optional: customFieldsOnly
UI02
URM sends status response and field definitions if no error
Returns: ermStatusCode, externalTableFields (RS) or ermStatusMessage, StatusCode, StatusMessage
UI03
Error?
UI04
Yes
Adapter evaluates status code and error message
UI05
No
Build update request
UI06
Call URM update service
EDIT_EXTERNAL
Requires: dSource, propertyList (localdata)
UI07
URM does update
UI08
URM sends status response
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
UI09
Adapter receives and evaluates status response
UI10

DECLARING OR UPDATING ITEMS IN URM

The adapter can also provide URM with item metadata and allow URM to determine whether the items should be declared or updated. The adapter does not need to know if the items exist already in URM; if the items do not exist URM performs a declaration (external checkin; the items are stored externally in the repository, not in URM), and if the items do exist (are being managed externally already) URM performs an update.

This section covers the following topics:

- Declaring or Updating Items in Batch (page 3-12)
- Declaring or Updating Individual Items (page 3-15)

Declaring or Updating Items in Batch

Items can be declared to URM (external checkin; the items are stored externally in the repository, not in URM) or have their metadata updated in batch. The batch declaration or update process is as follows (see Figure 3-3):

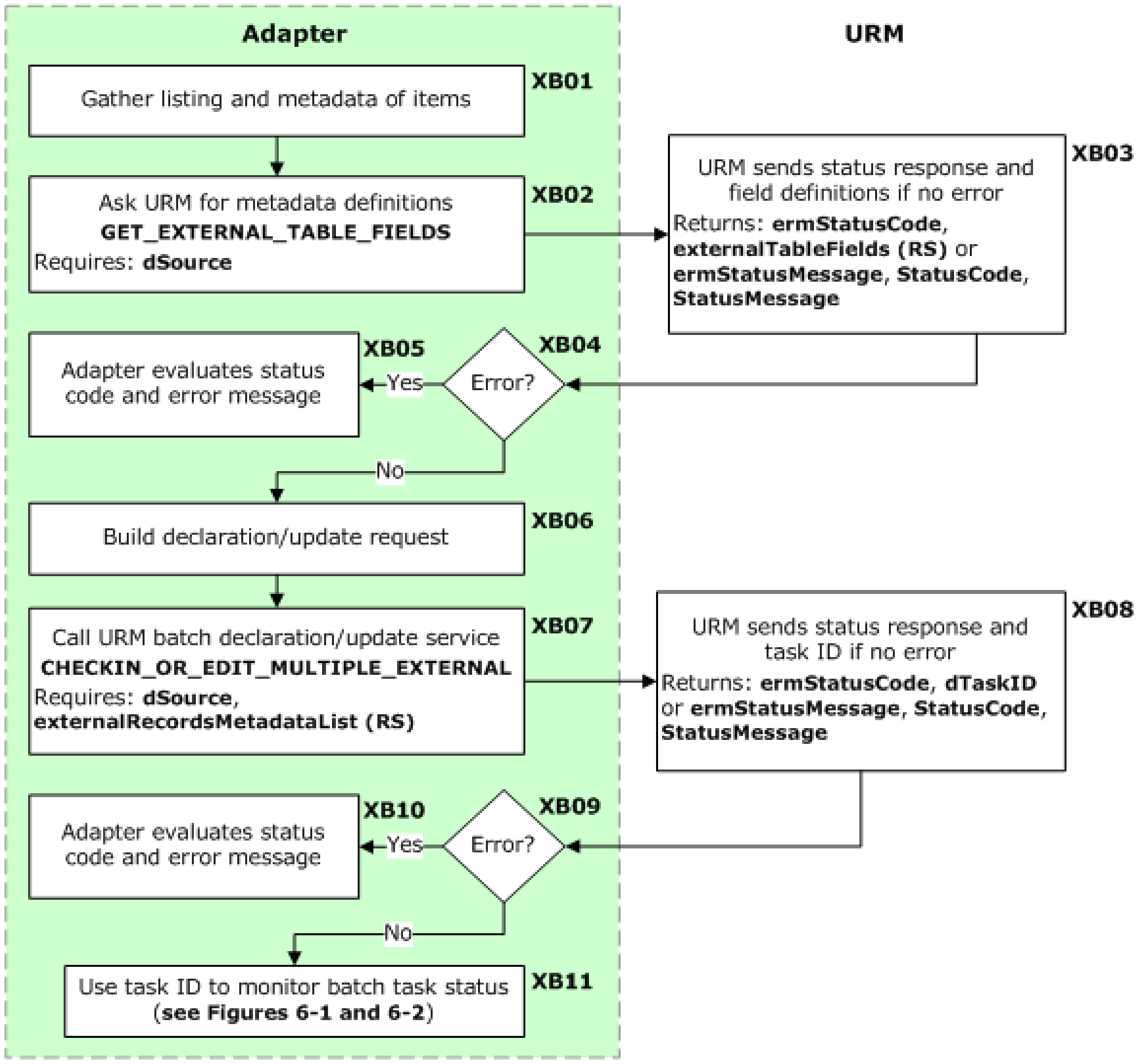
- **(XB01)** The adapter gathers a listing and the metadata of items to be declared or updated.
- **(XB02)** The adapter calls the GET_EXTERNAL_TABLE_FIELDS service to ask URM for metadata definitions so that the adapter knows how to format the metadata it is about to send to URM. For additional details, see GET_EXTERNAL_TABLE_FIELDS (page A-51).
- **(XB03)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the metadata definitions to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- **(XB04)** The adapter checks the status response.
- **(XB05)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(XB06)** If there is no error, the adapter receives the metadata definitions from URM. The adapter uses the metadata definitions and the listing and metadata it has collected to build a declaration/update request.

- ❖ **(XB07)** The adapter calls the CHECKIN_OR_EDIT_MULTIPLE_EXTERNAL service to declare the batch of items to URM or edit the batch of items in URM. Please note the following important considerations:
  - A record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error.
  - Neither dDocName nor dLongName can be updated.

  For additional details, see CHECKIN_OR_EDIT_MULTIPLE_EXTERNAL (page A-25).
- ❖ **(XB08)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the adapter a task ID. If there is an error, URM sends an error message (ermStatusMessage).

  When URM runs the batch job, it checks to see if each item exists in URM already. If the item does not exist, URM performs a declaration (external checkin). If the item exists, URM performs an update.
- ❖ **(XB09)** The adapter checks the status response.
- ❖ **(XB10)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- ❖ **(XB11)** If there is no error, the adapter receives a task ID from URM. The adapter uses the task ID to monitor the status of the batch task. For details, see Checking the Status of Individual Batch Tasks (page 6-2) and Checking the Status of Multiple Batch Tasks (page 6-5).

Figure 3-5 Declaring or updating items in batch


Adapter
URM
Gather listing and metadata of items
XB01
Ask URM for metadata definitions
GET_EXTERNAL_TABLE_FIELDS
Requires: dSource
XB02
URM sends status response and field definitions if no error
Returns: ermStatusCode, externalTableFields (RS) or ermStatusMessage, StatusCode, StatusMessage
XB03
Error?
XB04
Adapter evaluates status code and error message
XB05
Yes
No
Build declaration/update request
XB06
Call URM batch declaration/update service
CHECKIN_OR_EDIT_MULTIPLE_EXTERNAL
Requires: dSource, externalRecordsMetadataList (RS)
XB07
URM sends status response and task ID if no error
Returns: ermStatusCode, dTaskID or ermStatusMessage, StatusCode, StatusMessage
XB08
Error?
XB09
Adapter evaluates status code and error message
XB10
Yes
No
Use task ID to monitor batch task status (see Figures 6-1 and 6-2)
XB11

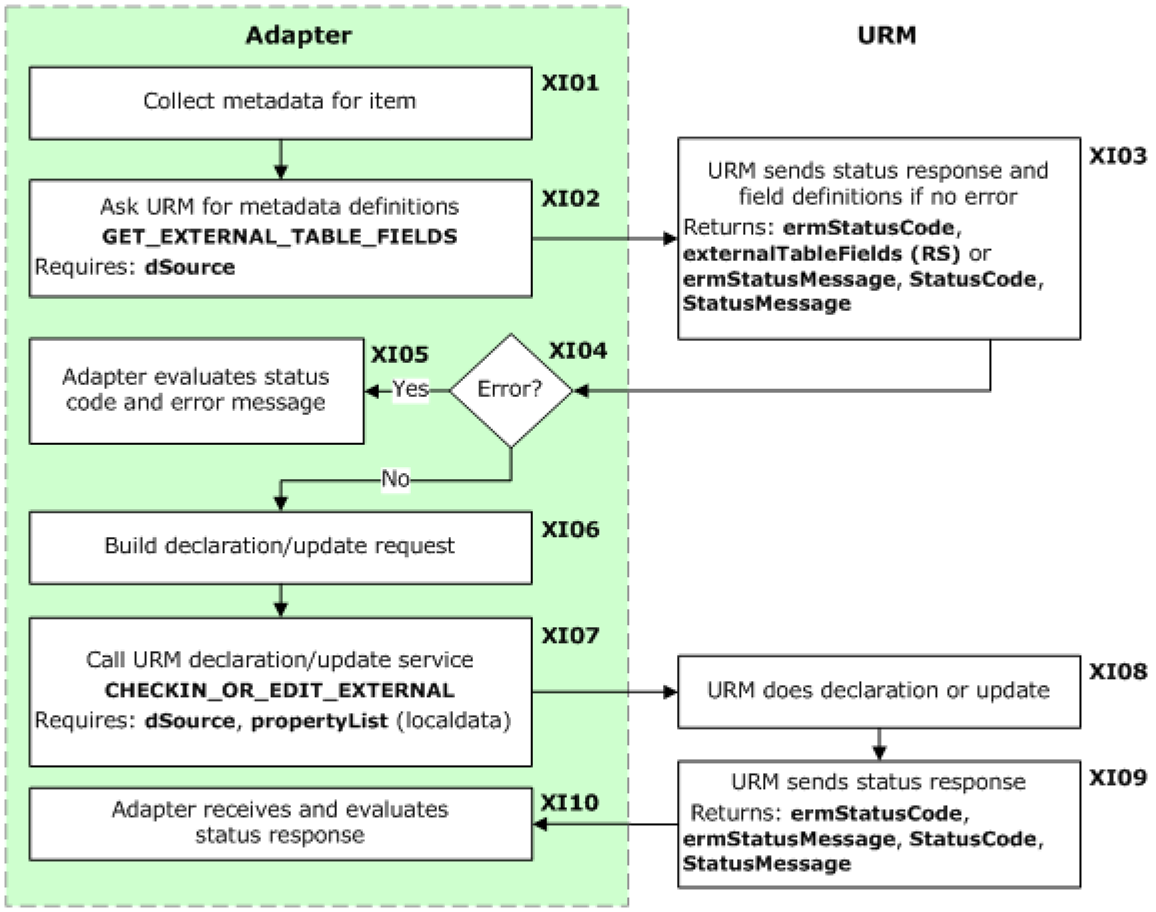
Declaring or Updating Individual Items

Items can be declared to URM (external checkin; the items are stored externally in the repository, not in URM) or have their metadata updated individually. The individual declaration or update process is as follows (see Figure 3-4):

- ❖ **(XI01)** The adapter collects metadata for the item to be declared or updated.
- ❖ **(XI02)** The adapter calls the GET_EXTERNAL_TABLE_FIELDS service to ask URM for metadata definitions so that the adapter knows how to format the metadata it is about to send to URM. For additional details, see GET_EXTERNAL_TABLE_FIELDS (page A-51).
- ❖ **(XI03)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the metadata definitions to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- ❖ **(XI04)** The adapter checks the status response.
- ❖ **(XI05)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- ❖ **(XI06)** If there is no error, the adapter receives the metadata definitions from URM. The adapter uses the metadata definitions and the metadata it has collected to build a declaration request.
- ❖ **(XI07)** The adapter calls the CHECKIN_OR_EDIT_EXTERNAL service to declare the item to URM or update the item in URM. Please note the following important considerations:
  - A record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error.
  - Neither dDocName nor dLongName can be updated.

  For additional details, see CHECKIN_OR_EDIT_EXTERNAL (page A-23).
- ❖ **(XI08)** If the item does not exist already, URM performs a declaration. If the item exists, URM performs an update.
- ❖ **(XI09)** URM sends a status response. The status response either indicates that the declaration or update was successful (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).
- ❖ **(XI10)** The adapter receives and evaluates the status response.

Figure 3-6 Declaring or updating individual items


Adapter
URM
Collect metadata for item
XI01
Ask URM for metadata definitions
GET_EXTERNAL_TABLE_FIELDS
Requires: dSource
XI02
URM sends status response and field definitions if no error
Returns: ermStatusCode, externalTableFields (RS) or ermStatusMessage, StatusCode, StatusMessage
XI03
Error?
XI04
Yes
No
Adapter evaluates status code and error message
XI05
Build declaration/update request
XI06
Call URM declaration/update service
CHECKIN_OR_EDIT_EXTERNAL
Requires: dSource, propertyList (localdata)
XI07
URM does declaration or update
XI08
URM sends status response
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
XI09
Adapter receives and evaluates status response
XI10

Deleting Items from URM

Non-record items within the repository might be deleted, either by users interacting with that repository or by the repository itself. Because URM holds metadata for the items, it is important that adapter inform URM of item deletions so that URM can stop tracking the retention lifecycle of the items.

This section covers the following topics:

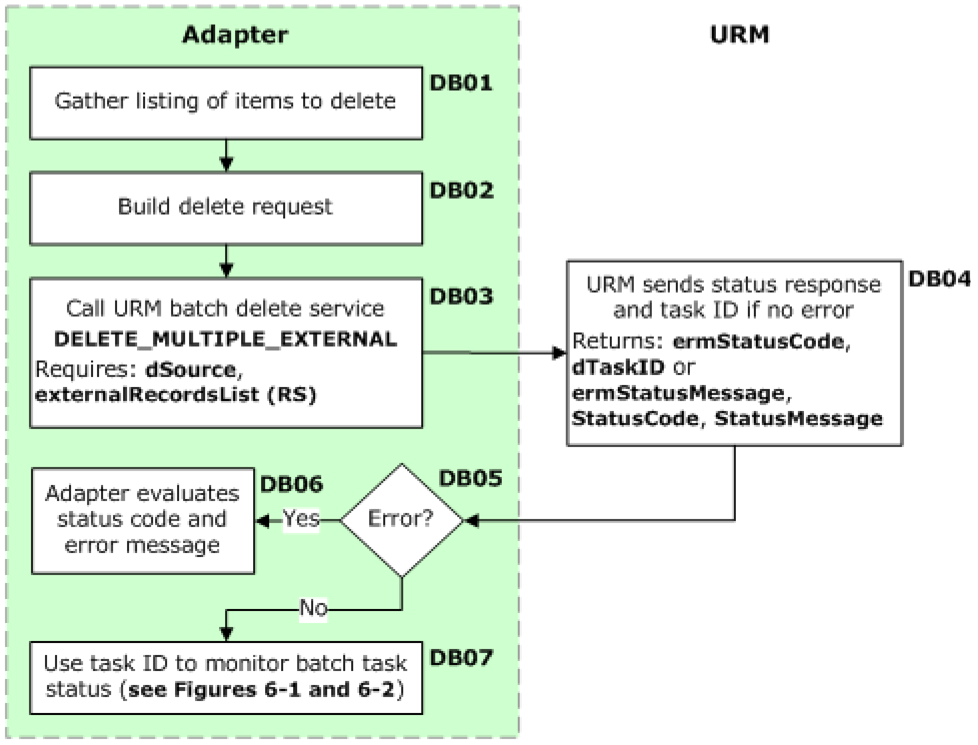
- Deleting Items in Batch (page 3-17)
- Deleting Individual Items (page 3-19)
- Deleting Items by Pattern (page 3-20)

Deleting Items in Batch

Items can be deleted from URM in a batch. The batch deletion process is as follows (see Figure 3-7):

- **(DB01)** The adapter gathers a listing of items to be deleted.
- **(DB02)** The adapter builds a delete request.
- **(UB07)** The adapter calls the DELETE_MULTIPLE_EXTERNAL service to delete the batch of items from URM. The adapter must provide a content server result set (externalRecordsList). The result set column must contain the adapter's key column. For additional details, see DELETE_MULTIPLE_EXTERNAL (page A-41).
- **(DB04)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the adapter a task ID. If there is an error, URM sends an error message (ermStatusMessage).
- **(DB05)** The adapter checks the status response.
- **(DB06)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(DB07)** If there is no error, the adapter receives a task ID from URM. The adapter uses the task ID to monitor the status of the batch deletion. For details, see Checking the Status of Individual Batch Tasks (page 6-2) and Checking the Status of Multiple Batch Tasks (page 6-5).

Figure 3-7 Deleting items in batch

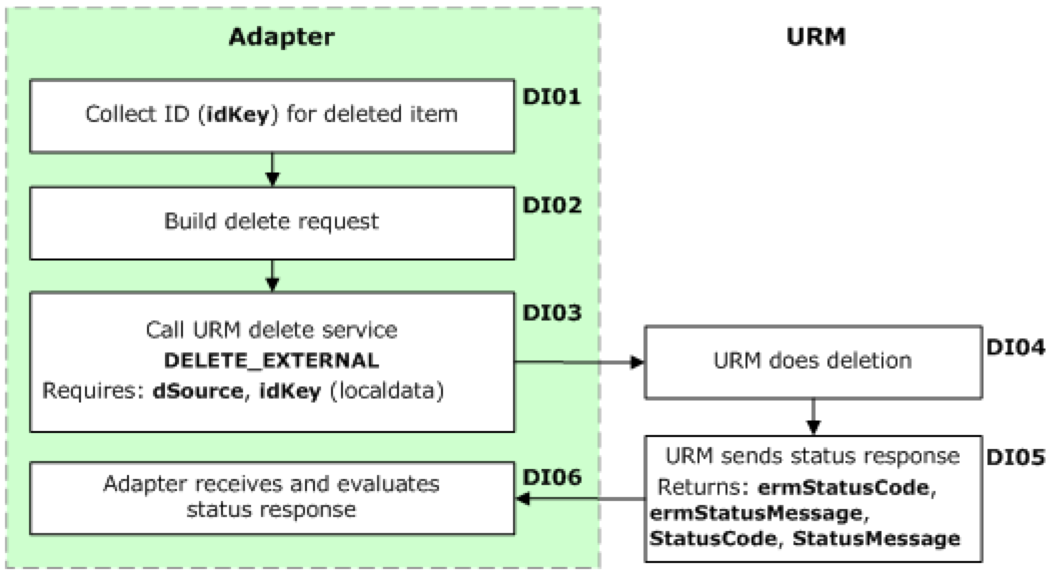

Adapter
URM
Gather listing of items to delete
DB01
Build delete request
DB02
Call URM batch delete service **DELETE_MULTIPLE_EXTERNAL** Requires: **dSource**, **externalRecordsList (RS)**
DB03
URM sends status response and task ID if no error Returns: **ermStatusCode**, **dTaskID** or **ermStatusMessage**, **StatusCode**, **StatusMessage**
DB04
Error?
DB05
Yes
Adapter evaluates status code and error message
DB06
No
Use task ID to monitor batch task status (**see Figures 6-1 and 6-2**)
DB07

Deleting Individual Items

Items can be deleted from URM individually. The individual deletion process is as follows (see Figure 3-8):

- **(DI01)** The adapter collects the ID (idKey) for the item to be deleted.
- **(DI02)** The adapter builds a delete request.
- **(DI03)** The adapter calls the DELETE_EXTERNAL service to delete the item from URM. For additional details, see DELETE_EXTERNAL (page A-40).
- **(DI04)** URM does the deletion.
- **(DI05)** URM sends a status response. The status response either indicates that the deletion was successful (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).
- **(DI06)** The adapter receives and evaluates the status response.

Figure 3-8 Deleting individual items


Adapter
URM
Collect ID (idKey) for deleted item
DI01
Build delete request
DI02
Call URM delete service
DELETE_EXTERNAL
Requires: dSource, idKey (localdata)
DI03
URM does deletion
DI04
URM sends status response
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
DI05
Adapter receives and evaluates status response
DI06

Deleting Items by Pattern

Items can be deleted from URM by pattern. URM deletes all items where the specified field starts with the specified pattern (assuming the items are not frozen).

If the optional dateField and dateValue are specified, all items that match the pattern criteria and are older than the date specified for the dateField will be deleted. Items that match the pattern criteria but are newer than the date specified will not be deleted.

The pattern deletion process is as follows (see Figure 3-9):

❖ **(DT01)** The adapter builds a pattern delete request.

❖ **(DT02)** The adapter calls the DELETE_BY_PATTERN service to delete the batch of items from URM. Please note the following important considerations:

  - The optional operator parameter applies to the field parameter. The dateField and dateValue parameters do not have an operator. The following operators are supported: beginsWith (default), endsWith, contains, equals. When the operator parameter is not passed, a beginsWith will always be performed.
  - If the optional dateField and dateValue are specified, all items that match the pattern criteria and are older than the date specified for the dateField will be deleted. Items that match the pattern criteria but are newer than the date specified will not be deleted. The dateValue is defined in the JDBC date format.

  For additional details, see DELETE_BY_PATTERN (page A-37).

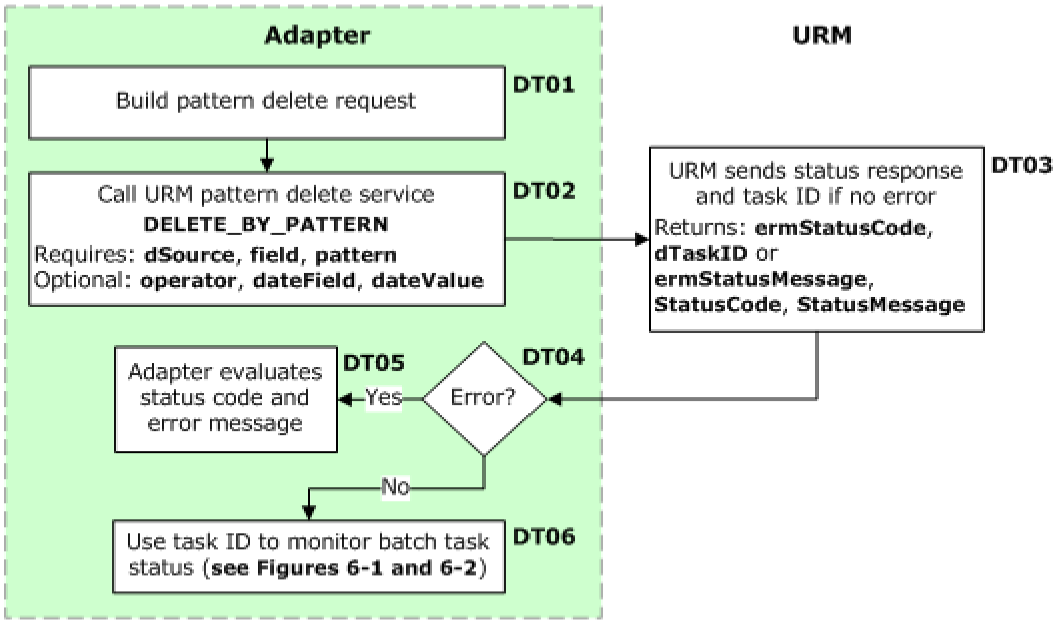
❖ **(DT03)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the adapter a task ID. If there is an error, URM sends an error message (ermStatusMessage).

❖ **(DT04)** The adapter checks the status response.

❖ **(DT05)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.

❖ **(DT06)** If there is no error, the adapter receives a task ID from URM. The adapter uses the task ID to monitor the status of the batch deletion. For details, see Checking the Status of Individual Batch Tasks (page 6-2) and Checking the Status of Multiple Batch Tasks (page 6-5).

Figure 3-9 Deleting items by pattern


Adapter
URM
Build pattern delete request
DT01
Call URM pattern delete service
DELETE_BY_PATTERN
Requires: dSource, field, pattern
Optional: operator, dateField, dateValue
DT02
URM sends status response and task ID if no error
Returns: ermStatusCode, dTaskID or ermStatusMessage, StatusCode, StatusMessage
DT03
Error?
DT04
Yes
Adapter evaluates status code and error message
DT05
No
Use task ID to monitor batch task status (see Figures 6-1 and 6-2)
DT06

CHECKING ITEMS INTO URM (INTERNAL CHECKIN)

Records can be preserved in place if the repository has the ability to ensure that the records will remain unalterable during the retention period. The repository also needs to be able to purge the records at the end of the retention period. For details, see Declaring Items to URM (External Checkin) (page 3-2) and Declaring or Updating Items in URM (page 3-12).

Certain repositories, such as file servers, might not be able to preserve records or non-records over their retention period or purge the records properly when it is time for disposal. Adapters associated with these repositories need to check the items into URM (internal checkin, where the items are moved into URM), so that the records can be preserved and purged properly.

The process for checking an item from the repository into the URM repository is as follows (see Figure 3-10):

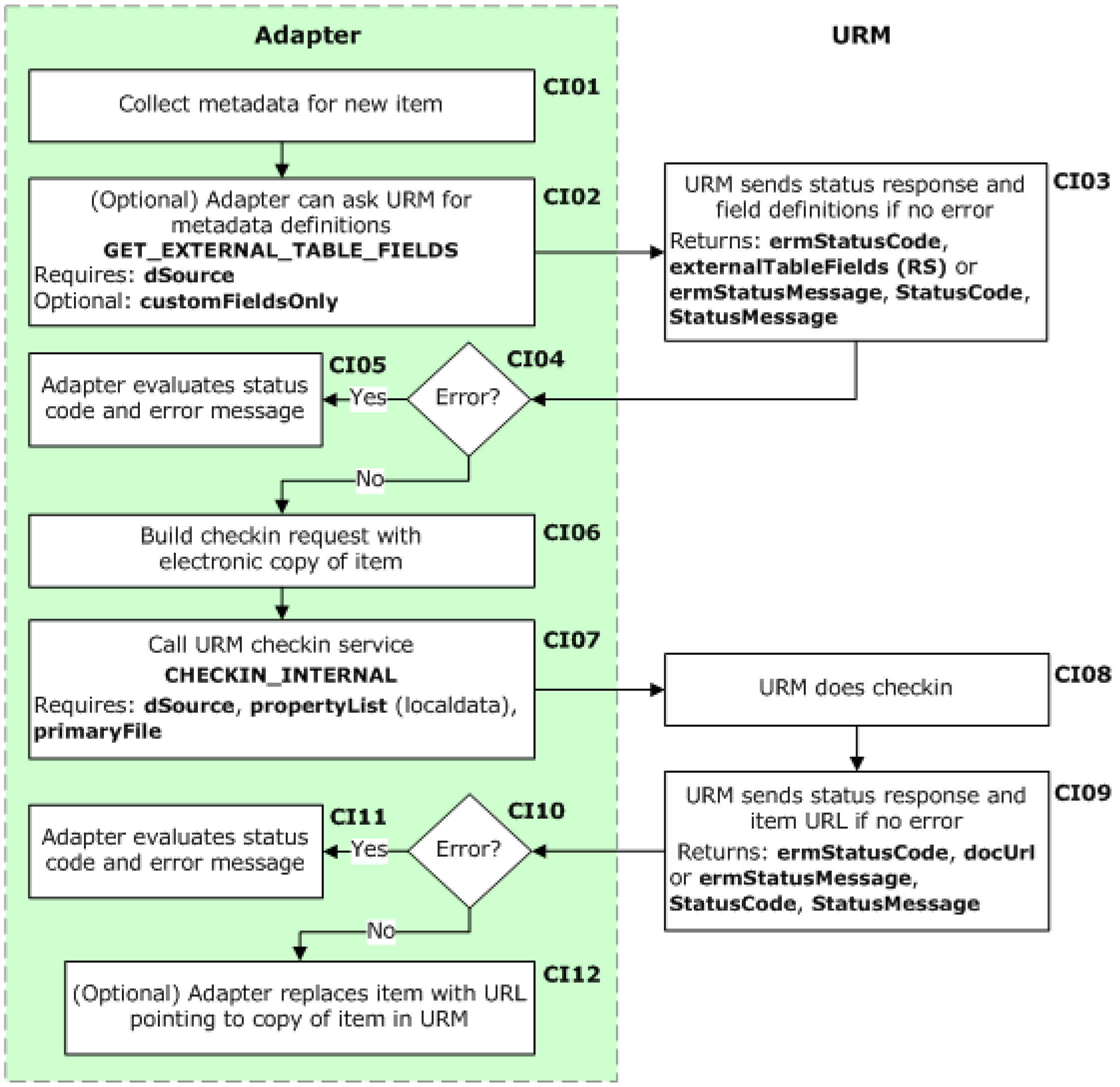
- **(CI01)** The adapter collects metadata for the item to be checked into URM.
- **(CI02)** If necessary, the adapter asks URM for metadata definitions so that the adapter knows how to format the metadata it is about to send to URM.
- **(CI03)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the metadata definitions to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- **(CI04)** The adapter checks the status response.
- **(CI05)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(CI06)** If there is no error, the adapter builds a checkin request that includes the item to be checked into URM.
- **(CI07)** The adapter calls the CHECKIN_INTERNAL service to check the item into URM (internal checkin). Please note the following considerations:
    - The dDocAuthor field is optional. If it is not passed, URM will use the authenticated user. The dDocAuthor field can be specified in the profile if you want to overwrite the value that is used for authentication.
    - If you want to use a profile to set values for fields, the profile must be set up in URM and you must map the xRMProfileTrigger field in the localdata that is passed to URM with this service.
    - The dDocType field must be set using a profile.

- The dDocName and dDocTitle fields are usually mapped fields that are handled outside of profiles.
- If the dSecurityGroup field is not mapped or passed, URM will set it to the default security group for the source or to Public.
- A record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error.
- The file to be checked into URM must be included.

For additional details, see CHECKIN_INTERNAL (page A-18).

- ❖ **(CI08)** URM does a checkin.
- ❖ **(CI09)** URM sends a status response. If there is no error (ermStatusCode=0), URM sends the URL for the item to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- ❖ **(CI10)** The adapter checks the status response.
- ❖ **(CI11)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- ❖ **(CI12)** If there is no error, optionally the adapter might then use the URL sent by URM to replace the item within the repository with a link to the copy of the item that is managed in URM. This enables users to still view the item if needed.

Figure 3-10 Checking in an item


Adapter
URM
Collect metadata for new item
CI01
(Optional) Adapter can ask URM for metadata definitions
GET_EXTERNAL_TABLE_FIELDS
Requires: dSource
Optional: customFieldsOnly
CI02
URM sends status response and field definitions if no error
Returns: ermStatusCode, externalTableFields (RS) or ermStatusMessage, StatusCode, StatusMessage
CI03
Adapter evaluates status code and error message
CI05
Yes
Error?
CI04
No
Build checkin request with electronic copy of item
CI06
Call URM checkin service
CHECKIN_INTERNAL
Requires: dSource, propertyList (localdata), primaryFile
CI07
URM does checkin
CI08
URM sends status response and item URL if no error
Returns: ermStatusCode, docUrl or ermStatusMessage, StatusCode, StatusMessage
CI09
Adapter evaluates status code and error message
CI11
Yes
Error?
CI10
No
(Optional) Adapter replaces item with URL pointing to copy of item in URM
CI12

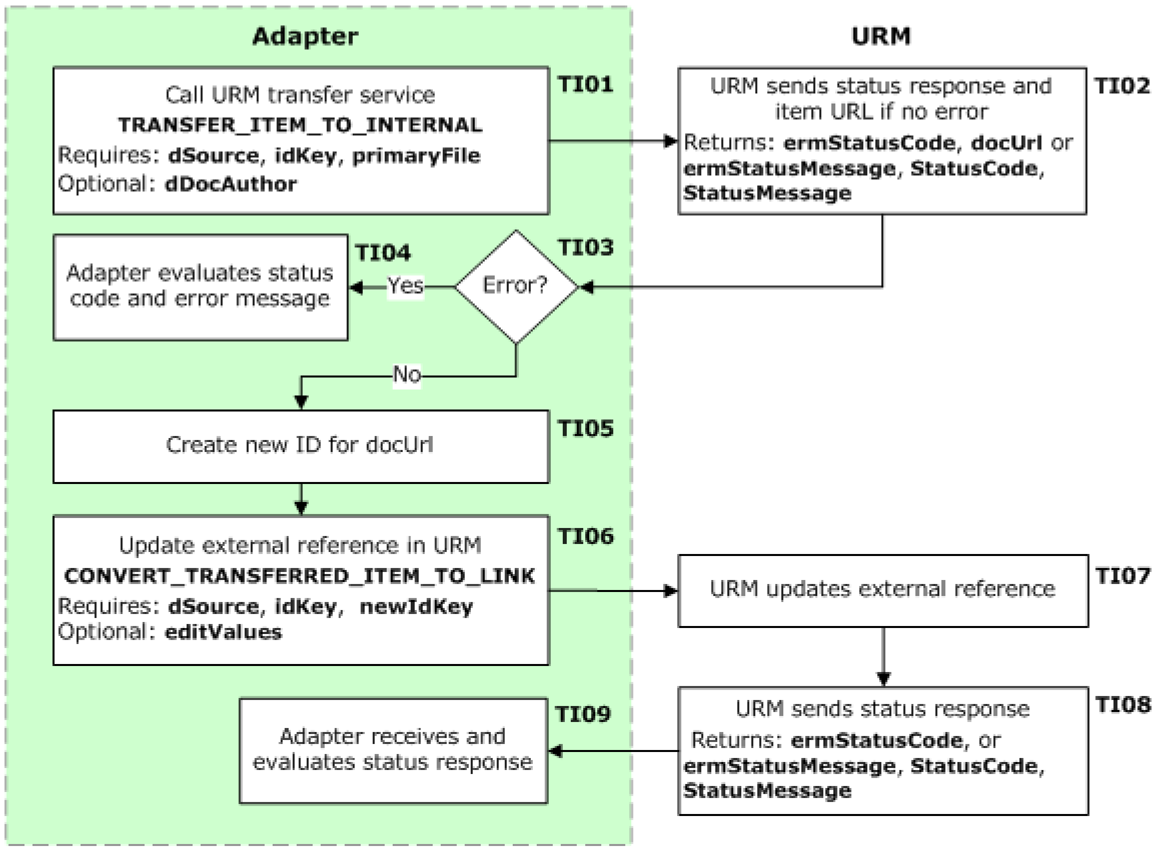
TRANSFERRING ITEMS TO URM

Items that are being managed externally by URM (have been declared to URM and are being stored externally in the repository) can be checked into URM (internal checkin, where the items are moved into the URM repository), so that the records can be preserved and purged properly. The process for transferring an item into the URM repository is as follows (see Figure 3-11):

- ❖ **(TI01)** The adapter calls the TRANSFER_ITEM_TO_INTERNAL service to transfer the item into URM. For additional details, see TRANSFER_ITEM_TO_INTERNAL (page A-83). Please note the following consideration:
  - dDocAuthor is optional. If dDocAuthor is sent, it cannot be the mapped value; it must be dDocAuthor. If dDocAuthor is not sent, URM will use the author value in the external table for the item to be transferred.
- ❖ **(TI02)** URM sends a status response. If there is no error (ermStatusCode=0), URM sends the URL for the item to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- ❖ **(TI03)** The adapter checks the status response.
- ❖ **(TI04)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- ❖ **(TI05)** If there is no error, the adapter uses the URL sent by URM to replace the original item with some sort of placeholder, and creates a new ID.
- ❖ **(TI06)** The adapter calls the CONVERT_TRANSFERRED_ITEM_TO_LINK service to update the external reference in URM with the new ID. URM does not delete the external reference, and if it is not updated it will point incorrectly to the original item instead of the placeholder. So, the adapter needs to update the external reference to point to the ID of the placeholder instead of the ID of the original item that is now stored in the URM repository. If you want to change any metadata values when the item is transferred to the URM repository, you can include the editValues parameter when calling this service. This propertyList only needs to include the metadata fields that should be changed. Any fields that are not specified will retain their original value. For additional details, see CONVERT_TRANSFERRED_ITEM_TO_LINK (page A-27).
- ❖ **(TI07)** URM updates the external reference with the new ID.

- **(TI08)** URM sends a status response. The status response either indicates that the update of the external reference was successful (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).
- **(TI09)** The adapter receives and evaluates the status response.

Figure 3-11 Transferring an item


Adapter
URM
Call URM transfer service
TRANSFER_ITEM_TO_INTERNAL
Requires: dSource, idKey, primaryFile
Optional: dDocAuthor
TI01
URM sends status response and item URL if no error
Returns: ermStatusCode, docUrl or ermStatusMessage, StatusCode, StatusMessage
TI02
Adapter evaluates status code and error message
TI04
Yes
Error?
TI03
No
Create new ID for docUrl
TI05
Update external reference in URM
CONVERT_TRANSFERRED_ITEM_TO_LINK
Requires: dSource, idKey, newIdKey
Optional: editValues
TI06
URM updates external reference
TI07
Adapter receives and evaluates status response
TI09
URM sends status response
Returns: ermStatusCode, or ermStatusMessage, StatusCode, StatusMessage
TI08

Chapter 4

Performing URM Tasks

Overview

This section covers the following topics:

Performing Federated Searches

As part of a discovery process, organizations might need to search through content across multiple repositories. URM can perform metadata searches internally using the metadata provided by adapters when declaring items. However, discovery requests often require that organizations do full text searches. The URM infrastructure performs full text searches by sending the search criteria to different adapters. The adapters check periodically to see if they have pending search criteria, perform the searches, and then upload the search resuts to URM.

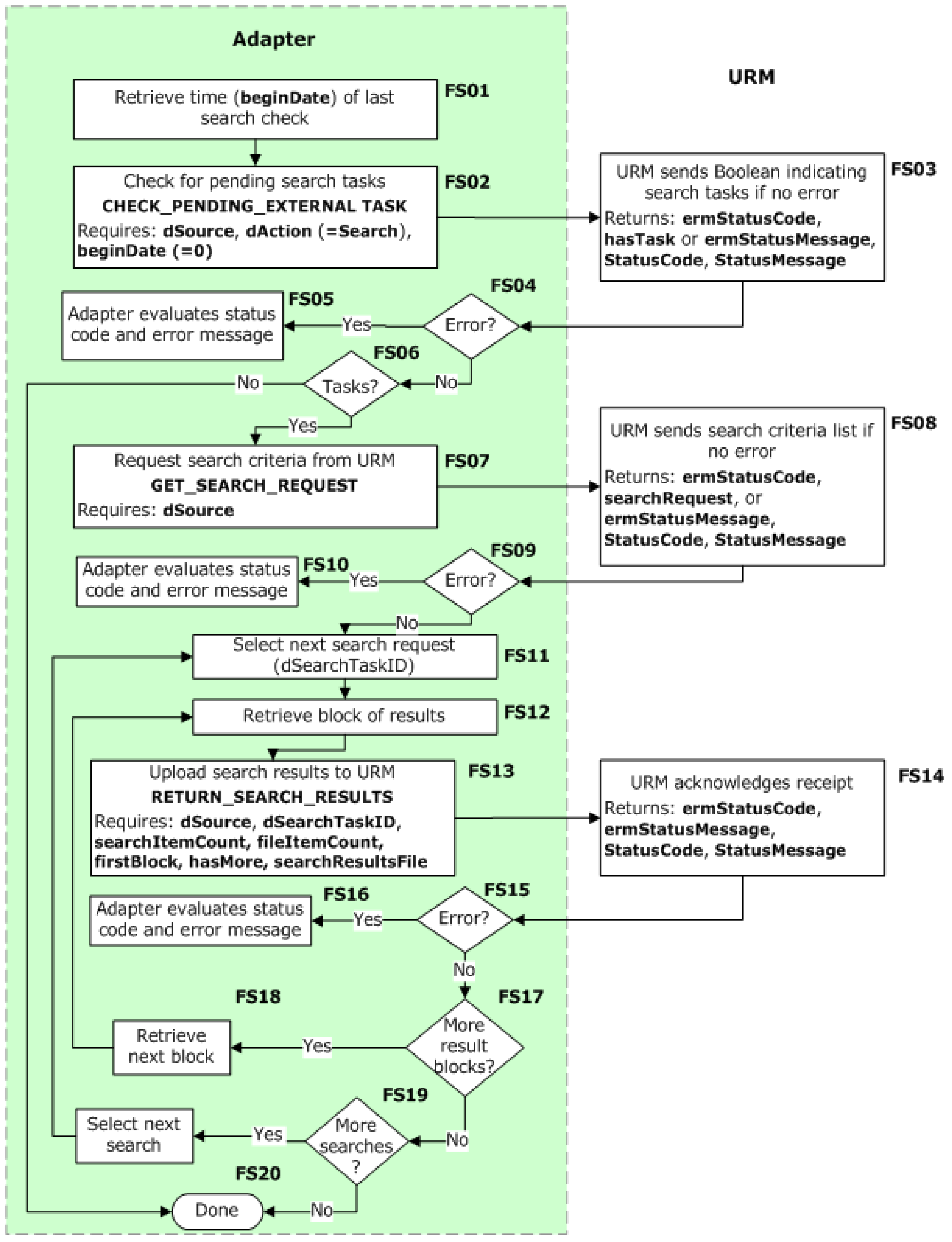
The federated search process is as follows (see Figure 4-1):

❖ **(FS01)** The adapter retrieves the time (beginDate) when it last processed searches. The adapter should leave beginDate empty if it is the first call to CHECK_PENDING_EXTERNAL_TASK.

- **(FS02)** The adapter calls the CHECK_PENDING_EXTERNAL_TASK service to check URM for pending search tasks that the adapter has not performed. As part of this check, the adapter sends the time when it last processed searches. For additional details, see CHECK_PENDING_EXTERNAL_TASK (page A-11).
- **(FS03)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends a Boolean flag (hasTask) indicating whether or not there are pending searches for the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- **(FS04)** The adapter checks the status response.
- **(FS05)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(FS06)** If there is no error, the adapter checks to see if there are pending searches. If there are no pending searches (hasTask=false), the adapter stops search processing.
- **(FS07)** If there are pending searches (hasTask=true), the adapter calls the GET_SEARCH_REQUEST service to request a list of pending search criteria from URM. For additional details, see GET_SEARCH_REQUEST (page A-61).
- **(FS08)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends a list of pending search criteria to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- **(FS09)** The adapter checks the status response.
- **(FS10)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(FS11)** If there is no error, the adapter starts processing the pending searches, selecting the first search criteria in the listing.
- **(FS12)** The adapter runs the search against the repository based on that criteria. The results may be divided into blocks in cases that there are a large amount of files to upload.
- **(FS13)** The adapter calls the RETURN_SEARCH_RESULTS service to upload the search result block to URM. For additional details, see RETURN_SEARCH_RESULTS (page A-77).
- **(FS14)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM acknowledges the receipt of the search results. If there is an error, URM sends an error message (ermStatusMessage).
- **(FS15)** The adapter checks the status response.

- **(FS16)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(FS17)** The adapter checks to see if there are remaining result blocks for the search criteria.
- **(FS18)** If there are more result sets, the adapter selects the next block of results and repeats the search process (FS12–FS18).
- **(FS19)** If there is no error, the adapter checks to see if there are more search requests pending. If there are more searches, the adapter gets the next search criteria in the list and repeats the search process (FS11-FS18).
- **(FS20)** If there are no more searches, the adapter stops search processing.

Figure 4-1 Searching for items


Adapter
URM
Retrieve time (beginDate) of last search check
FS01
Check for pending search tasks
CHECK_PENDING_EXTERNAL TASK
Requires: dSource, dAction (=Search), beginDate (=0)
FS02
URM sends Boolean indicating search tasks if no error
Returns: ermStatusCode, hasTask or ermStatusMessage, StatusCode, StatusMessage
FS03
FS04
Error?
Yes
Adapter evaluates status code and error message
FS05
No
Tasks?
FS06
No
Yes
Request search criteria from URM
GET_SEARCH_REQUEST
Requires: dSource
FS07
URM sends search criteria list if no error
Returns: ermStatusCode, searchRequest, or ermStatusMessage, StatusCode, StatusMessage
FS08
FS09
Error?
Yes
Adapter evaluates status code and error message
FS10
No
Select next search request (dSearchTaskID)
FS11
Retrieve block of results
FS12
Upload search results to URM
RETURN_SEARCH_RESULTS
Requires: dSource, dSearchTaskID, searchItemCount, fileItemCount, firstBlock, hasMore, searchResultsFile
FS13
URM acknowledges receipt
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
FS14
FS15
Error?
Yes
Adapter evaluates status code and error message
FS16
No
FS17
More result blocks?
Yes
Retrieve next block
FS18
No
FS19
More searches ?
Yes
Select next search
FS20
No
Done

Performing Dispositions

At the end of the retention period for a set of items stored in a repository, URM needs to direct the corresponding adapter to dispose of the items. URM directs the adapter to do so when the adapter checks to see if there are any dispositions for it to perform. Adapters should be configured to perform that check periodically.

The disposition process is as follows (see Figure 4-1):

- **(DA01)** The adapter determines the time when it last processed dispositions.

  If the adapter has not processed dispositions before, it should leave beginDate empty.

  If the adapter has processed dispositions before, it should have saved the endDate it received from URM when it called the LIST_EXTERNAL_APPROVED_DISP_ACTIONS service (DA07). The adapter must use this endDate as the beginDate for the next call to the CHECK_PENDING_EXTERNAL_TASK service (DA02) and the LIST_EXTERNAL_APPROVED_DISP_ACTIONS service (DA07).

Note: The beginDate and endDate are defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).

- **(DA02)** The adapter calls the CHECK_PENDING_EXTERNAL_TASK service to check URM for pending approved dispositions that the adapter has not performed. For additional details, see CHECK_PENDING_EXTERNAL_TASK (page A-11).
- **(DA03)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends a Boolean flag (hasTask) indicating whether or not there are any pending approved dispositions for the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- **(DA04)** The adapter checks the status response.
- **(DA05)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(DA06)** If there is no error, the adapter checks to see if there are any pending approved dispositions. If there are no pending approved dispositions (`hasTask`=false), the adapter stops disposition processing.
- **(DA07)** If there are pending approved dispositions (hasTask=true), the adapter calls the LIST_EXTERNAL_APPROVED_DISP_ACTIONS service to request a list of pending approved dispositions from URM. For additional details, see

LIST_EXTERNAL_APPROVED_DISP_ACTIONS (page A-65). Please note the following important considerations:

- When calling the LIST_EXTERNAL_APPROVED_DISP_ACTIONS service, the adapter must include the beginDate that it has determined (DA01). If there is no error with the service request, URM will return an endDate along with the approvedDispActionsList result set. The endDate identifies the time that the disposition query ran. The adapter must save this endDate, as the adapter must use this endDate as the beginDate the next time it calls the CHECK_PENDING_EXTERNAL_TASK and LIST_EXTERNAL_APPROVED_DISP_ACTIONS services.
- The beginDate and endDate are defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).

❖ **(DA08)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends a list of pending approved dispositions and an endDate to the adapter. If there is an error, URM sends an error message (ermStatusMessage).

❖ **(DA09)** The adapter checks the status response.

❖ **(DA10)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.

❖ **(DA11)** If there is no error, the adapter starts processing the pending approved dispositions, selecting the first disposition in the listing.

Note: The adapter must save the endDate that is returned, as the adapter must use this endDate as the beginDate the next time it calls the CHECK_PENDING_EXTERNAL_TASK service (DA02) and the LIST_EXTERNAL_APPROVED_DISP_ACTIONS service (DA07).

❖ **(DA12)** The adapter calls the LIST_EXTERNAL_ITEMS_FOR_DISP_ACTION service to request a list of items subject to that dispostion from URM. For additional details, see LIST_EXTERNAL_ITEMS_FOR_DISP_ACTION (page A-69).

❖ **(DA13)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the list of items to the adapter and a Boolean flag (hasMore) indicating if there are more items for that disposition. If there is an error, URM sends an error message (ermStatusMessage).

❖ **(DA14)** The adapter checks the status response.

❖ **(DA15)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.

❖ **(DA16)** If there is no error, the adapter checks to see if the action is an archive action. URM will return dDispAction=wwRmaArchive for the following action types:

- Accession
- Archive
- Move
- Transfer

❖ **(DA17)** If the action is an archive action (dDispAction=wwRmaArchive), the adapter bundles the corresponding items into an archive zip file and calls the UPLOAD_EXTERNAL_ARCHIVE service. The zip file of items must contain an *index.hda* file. For additional details, see UPLOAD_EXTERNAL_ARCHIVE (page A-92). If the action is not an archive action, skip to DA22.

❖ **(DA18)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the adapter a task ID. If there is an error, URM sends an error message (ermStatusMessage).

URM runs the upload job and marks the disposition as pending. The URM administrator then needs to evaluation the results and take manual actions to finish the task and mark it complete, so that the adapter (which is monitoring the task) can continue with the disposition process (DA21).

❖ **(DA19)** The adapter checks the status response.

❖ **(DA20)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.

❖ **(DA21)** If there is no error, the adapter receives a task ID from URM. The adapter uses the task ID to monitor the status of the archive upload task. When the task is complete, the adapter checks to see if there are more items for the disposition (DA28).

❖ **(DA22)** If the action is not an archive action, the adapter disposes of the items in the list.

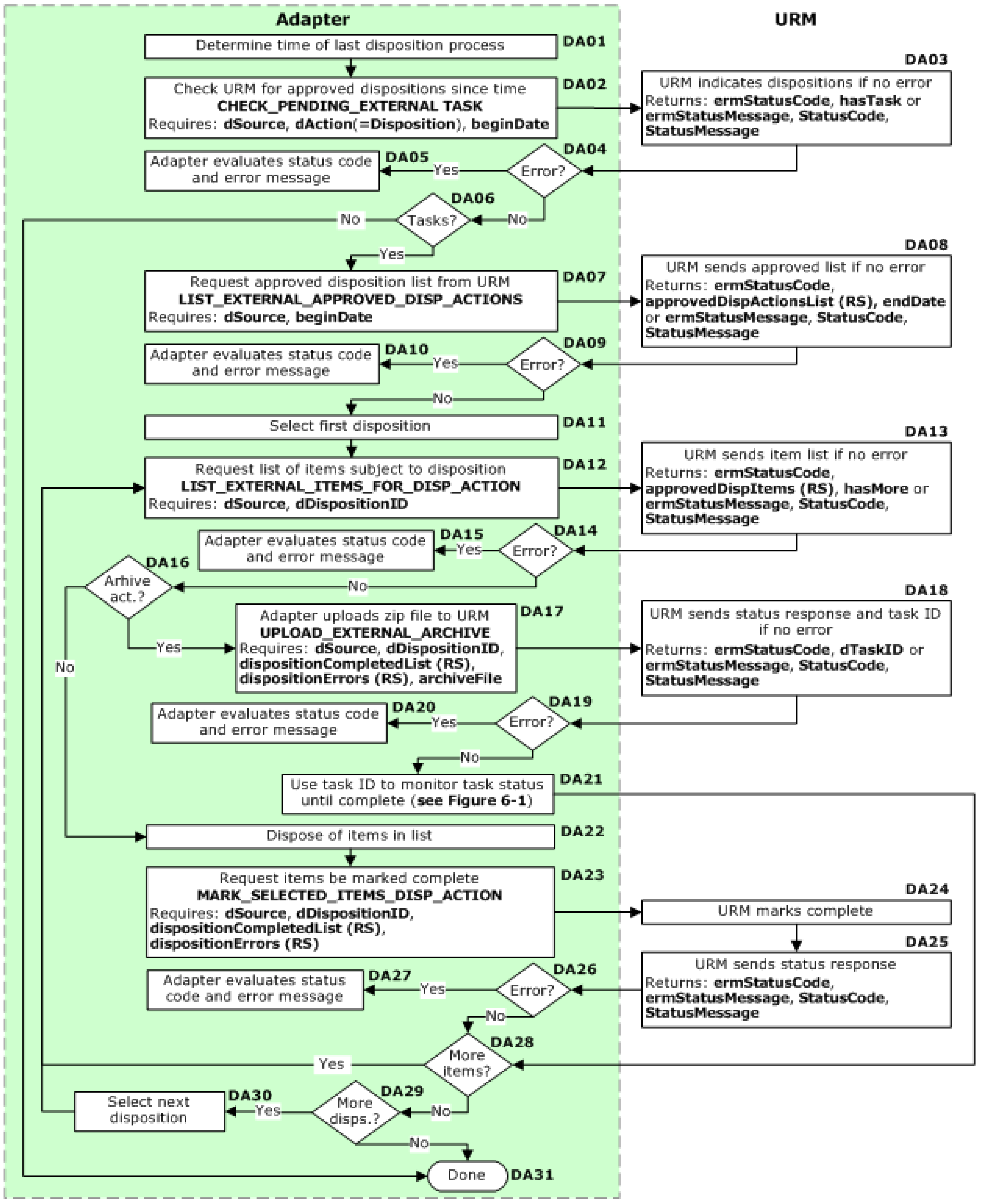
❖ **(DA23)** The adapter calls the MARK_SELECTED_ITEMS_DISP_ACTION service to request URM to mark the disposition of the items as complete. For additional details, see MARK_SELECTED_ITEMS_DISP_ACTION (page A-71).

❖ **(DA24)** URM marks the disposition of the items as complete.

Note: In the URM user interface, the Transfer action will remain in the My Completion List for <external_source_name> until the URM administrator downloads the archive zip file and marks all archived items in the zip file as complete or addresses any errors.

- **(DA25)** URM sends a status response to the adapter. The status response either indicates that the disposition of the items has been marked as complete (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).
- **(DA26)** The adapter checks the status response.
- **(DA27)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(DA28)** If there is no error, the adapter checks to see if there are more items for the disposition (checks hasMore Boolean flag, returned in DA13). If there are more items for the disposition, the adapter requests the next set of items and repeats the process on the next set of items (DA12–DA28).
- **(DA29)** If there are no more items for the disposition, the adapter checks to see if there are more pending approved dispositions (checks approvedDispActionsList result set, returned in DA08).
- **(DA30)** If there are more pending approved dispositions, the adapter selects the next disposition in the list and repeats the process for the next disposition (DA12–DA29).
- **(DA31)** If there are no more pending approved dispositions, the adapter stops disposition processing.

Figure 4-1 Performing dispositions


Adapter
URM
Determine time of last disposition process
DA01
Check URM for approved dispositions since time
CHECK_PENDING_EXTERNAL TASK
Requires: dSource, dAction(=Disposition), beginDate
DA02
DA03
URM indicates dispositions if no error
Returns: ermStatusCode, hasTask or ermStatusMessage, StatusCode, StatusMessage
Adapter evaluates status code and error message
DA05
Yes
Error?
DA04
No
Tasks?
DA06
No
Yes
Request approved disposition list from URM
LIST_EXTERNAL_APPROVED_DISP_ACTIONS
Requires: dSource, beginDate
DA07
DA08
URM sends approved list if no error
Returns: ermStatusCode, approvedDispActionsList (RS), endDate or ermStatusMessage, StatusCode, StatusMessage
Adapter evaluates status code and error message
DA10
Yes
Error?
DA09
No
Select first disposition
DA11
Request list of items subject to disposition
LIST_EXTERNAL_ITEMS_FOR_DISP_ACTION
Requires: dSource, dDispositionID
DA12
DA13
URM sends item list if no error
Returns: ermStatusCode, approvedDispItems (RS), hasMore or ermStatusMessage, StatusCode, StatusMessage
Adapter evaluates status code and error message
DA15
Yes
Error?
DA14
DA16
Arhive act.?
No
Yes
Adapter uploads zip file to URM
UPLOAD_EXTERNAL_ARCHIVE
Requires: dSource, dDispositionID, dispositionCompletedList (RS), dispositionErrors (RS), archiveFile
DA17
DA18
URM sends status response and task ID if no error
Returns: ermStatusCode, dTaskID or ermStatusMessage, StatusCode, StatusMessage
No
Adapter evaluates status code and error message
DA20
Yes
Error?
DA19
No
Use task ID to monitor task status until complete (see Figure 6-1)
DA21
Dispose of items in list
DA22
Request items be marked complete
MARK_SELECTED_ITEMS_DISP_ACTION
Requires: dSource, dDispositionID, dispositionCompletedList (RS), dispositionErrors (RS)
DA23
DA24
URM marks complete
DA25
URM sends status response
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
Adapter evaluates status code and error message
DA27
Yes
Error?
DA26
No
More items?
DA28
Yes
Select next disposition
DA30
Yes
More disps.?
DA29
No
No
Done
DA31

Performing Holds/Freezes

As part of the discovery process, organizations might need to place a litigation or audit hold (or “freeze”) on certain items. The URM infrastructure should keep those items from being edited or deleted. For items stored within external repositories, URM relies on adapters to communicate the litigation hold status of the items to the repository so that the items can be preserved properly. URM relays the list of items on hold to the adapter when the adapter checks to see if any items have been recently placed on hold. Adapters should be configured to perform that check periodically.

The hold/freeze process is as follows (see Figure 4-2):

- **(HF01)** The adapter determines the time when it last processed freezes.

  If the adapter has not processed freezes before, it should leave beginDate empty.

  If the adapter has processed freezes before, it should have saved the endDate it received from URM the last time it called the GET_EXTERNAL_FREEZE_LIST service (HF07). The adapter must use this endDate as the beginDate for the next call to the CHECK_PENDING_EXTERNAL_TASK service (HF02) and the GET_EXTERNAL_FREEZE_LIST service (HF07).

Note: The beginDate and endDate are defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).

- **(HF02)** The adapter calls the CHECK_PENDING_EXTERNAL_TASK service to check URM for new freezes. For additional details, see CHECK_PENDING_EXTERNAL_TASK (page A-11).
- **(HF03)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends a Boolean flag (hasTask) indicating whether or not there are new freezes for the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- **(HF04)** The adapter checks the status response.
- **(HF05)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(HF06)** If there is no error, the adapter checks to see if there are new freezes. If there are no new freezes (hasTask=false), the adapter stops freeze processing.
- **(HF07)** If there are new freezes (hasTask=true), the adapter calls the GET_EXTERNAL_FREEZE_LIST service to request a list of frozen items from

URM. For additional details, see GET_EXTERNAL_FREEZE_LIST (page A-48). Please note the following important considerations:

- When calling the GET_EXTERNAL_FREEZE_LIST service, the adapter must include the beginDate that it has determined (HF01). If there is no error with the service request, URM will return an externalFreezeList result set listing a block of frozen items, an endDate identifying when the query ran, and a Boolean flag (hasMore) indicating if there are more freeze items.

  When the adapter is finished processing the block of items, it uses the endDate that was sent with the block of items as the retrieveDate to mark the items as frozen.If there are more freeze items, the adapter calls the GET_EXTERNAL_FREEZE_LIST service again, using the same beginDate as it did initially. The adapter then receives the next block of items and a new corresponding endDate, which becomes the retrieveDate for this next block of items.

  When there are no more freeze items, the adapter must save the endDate for the last block of items received, as the adapter must use this endDate as the beginDate the next time it calls the CHECK_PENDING_EXTERNAL_TASK and GET_EXTERNAL_FREEZE_LIST services.
- The beginDate and endDate are defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).

❖ **(HF08)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the list of frozen items to the adapter, an endDate for the block of items, and a Boolean flag (hasMore) indicating if there are more frozen items. If there is an error, URM sends an error message (ermStatusMessage).

❖ **(HF09)** The adapter checks the status response.

❖ **(HF10)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.

❖ **(HF11)** If there is no error, the adapter starts processing the frozen items, selecting the first item on the list.

❖ **(HF12)** The adapter tags the item as frozen in the repository. If the repository cannot keep the item from being edited or deleted, the adapter might create a "hold" copy of the item within a part of the repository that users cannot reach to edit the item. It might relay the location of that hold copy to URM by performing the item update process to specify that hold location as a metadata value. For details on the update process, see Updating Items in URM (page 3-7).

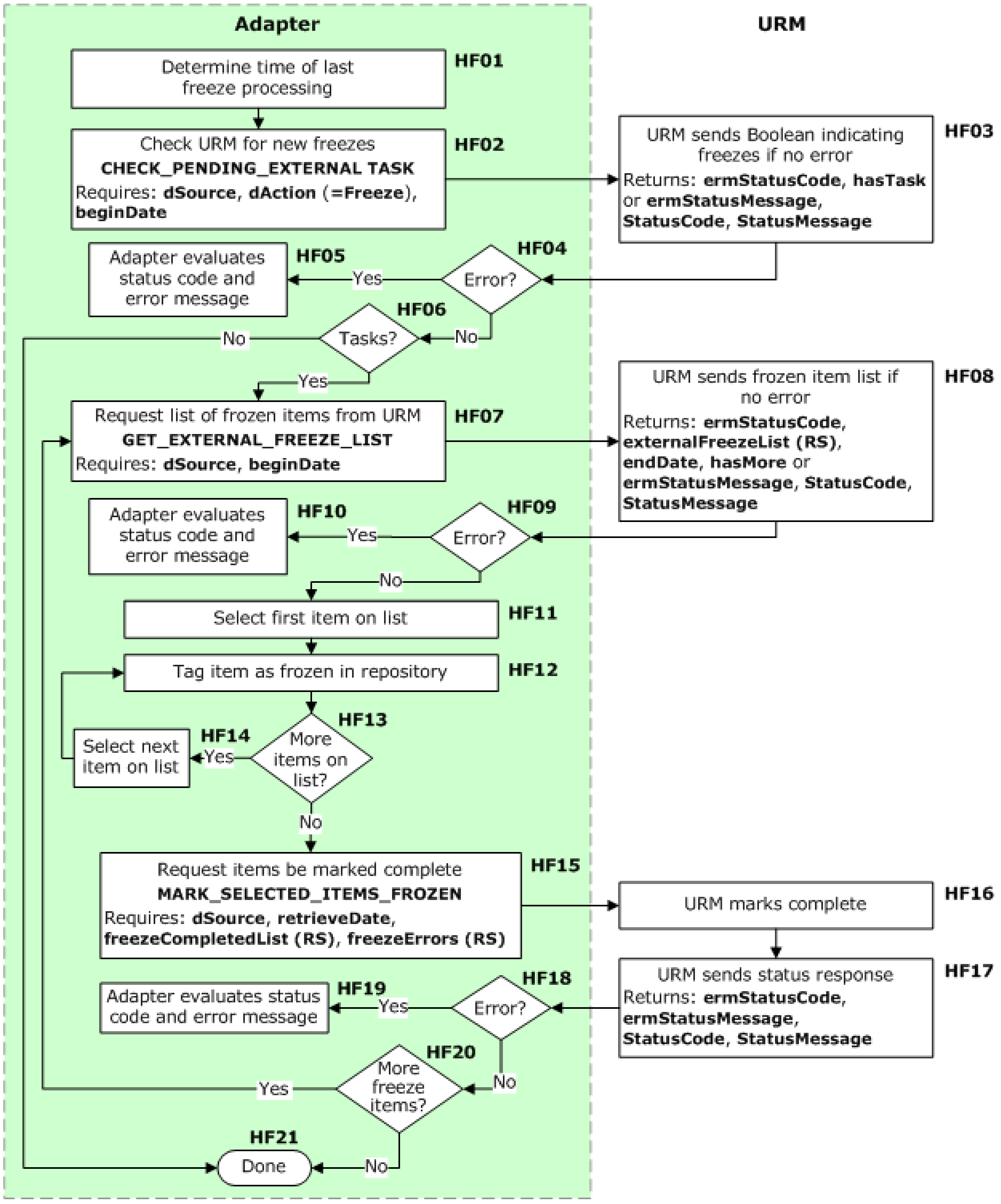
- **(HF13)** The adapter checks to see if there are more items on the list.
- **(HF14)** If there are more items on the list, the adapter selects the next item on the list and repeats the freeze process (HF11–HF13).
- **(HF15)** If there are no more items on the list, the adapter calls the MARK_SELECTED_ITEMS_FROZEN service to request URM to mark the freeze of the items as complete. For additional details, see MARK_SELECTED_ITEMS_FROZEN (page A-72).

Important: When calling the MARK_SELECTED_ITEMS_FROZEN service, you must include the retrieveDate parameter. It is critical that the retrieveDate match, exactly, the endDate that was returned by URM for the corresponding block of items when the GET_EXTERNAL_FREEZE_LIST service was called (HF07–HF08). The retrieveDate is defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).

- **(HF16)** URM marks the freeze of the items as complete.
- **(HF17)** URM sends a status response to the adapter. The status response either indicates that the freeze of the items has been marked as complete (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).
- **(HF18)** The adapter checks the status response.
- **(HF19)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(HF20)** If there is no error, the adapter checks to see if there are more freeze items (checks hasMore Boolean flag, returned in HF08). If there are more freeze items, the adapter requests the next set of items and repeats the freeze process (HF12–HF20).
- **(HF21)** If there are no more freeze items, the adapter stops freeze processing.

Note: When there are no more freeze items, the adapter must save the endDate for the last block of items received, as the adapter must use this endDate as the beginDate the next time it calls the CHECK_PENDING_EXTERNAL_TASK service (HF02) and the GET_EXTERNAL_FREEZE_LIST service (HF07).

Figure 4-2 Performing a hold/freeze


Adapter
URM
Determine time of last freeze processing
HF01
Check URM for new freezes
CHECK_PENDING_EXTERNAL TASK
Requires: dSource, dAction (=Freeze), beginDate
HF02
URM sends Boolean indicating freezes if no error
Returns: ermStatusCode, hasTask or ermStatusMessage, StatusCode, StatusMessage
HF03
Adapter evaluates status code and error message
HF05
Yes
Error?
HF04
HF06
No
Tasks?
No
Yes
Request list of frozen items from URM
GET_EXTERNAL_FREEZE_LIST
Requires: dSource, beginDate
HF07
URM sends frozen item list if no error
Returns: ermStatusCode, externalFreezeList (RS), endDate, hasMore or ermStatusMessage, StatusCode, StatusMessage
HF08
Adapter evaluates status code and error message
HF10
Yes
Error?
HF09
No
Select first item on list
HF11
Tag item as frozen in repository
HF12
HF13
More items on list?
HF14
Select next item on list
Yes
No
Request items be marked complete
MARK_SELECTED_ITEMS_FROZEN
Requires: dSource, retrieveDate, freezeCompletedList (RS), freezeErrors (RS)
HF15
URM marks complete
HF16
URM sends status response
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
HF17
Adapter evaluates status code and error message
HF19
Yes
Error?
HF18
HF20
More freeze items?
No
Yes
HF21
Done
No

REMOVING HOLDS/FREEZES

When a litigation or audit hold (or “freeze”) is removed, items that had previously been marked with that hold need to be unmarked so that they can be edited or deleted normally. URM relies on adapters to communicate the removal of litigation hold on the items to the repository. URM relays the list of items for which a hold or freeze has been removed to the adapter when the adapter checks to see if any items have been had holds removed from them. Adapters should be configured to perform that check periodically.

The hold/freeze removal process is as follows (see Figure 4-3):

❖ **(RH01)** The adapter determines the time when it last processed unfreezes.

  If the adapter has not processed unfreezes before, it should leave beginDate empty.

  If the adapter has processed unfreezes before, it should have saved the endDate it received from URM the last time it called the GET_EXTERNAL_UNFREEZE_LIST service (RH07). The adapter must use this endDate as the beginDate for the next call to the CHECK_PENDING_EXTERNAL_TASK service (RH02) and the GET_EXTERNAL_UNFREEZE_LIST service (RH07).

Note: The beginDate and endDate are defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).

❖ **(RH02)** The adapter calls the CHECK_PENDING_EXTERNAL_TASK service to check URM for new unfreezes. For additional details, see CHECK_PENDING_EXTERNAL_TASK (page A-11).

❖ **(RH03)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends a Boolean flag (hasTask) indicating whether or not there are new unfreezes for the adapter. If there is an error, URM sends an error message (ermStatusMessage).

❖ **(RH04)** The adapter checks the status response.

❖ **(RH05)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.

❖ **(RH06)** If there is no error, the adapter checks to see if there are new unfreezes. If there are no new unfreezes (hasTask=false), the adapter stops unfreeze processing.

❖ **(RH07)** If there are new unfreezes (hasTask=true), the adapter calls the GET_EXTERNAL_UNFREEZE_LIST service to request a list of unfrozen items from URM. For additional details, see GET_EXTERNAL_UNFREEZE_LIST (page A-53). Please note the following important considerations:

- When calling the GET_EXTERNAL_UNFREEZE_LIST service, the adapter must include the beginDate that it has determined (RH01). If there is no error with the service request, URM will return an externalUnFreezeList result set listing a block of unfrozen items, an endDate identifying when the query ran, and a Boolean flag (hasMore) indicating if there are more unfreeze items.

  When the adapter is finished processing the block of items, it uses the endDate that was sent with the block of items as the retrieveDate to mark the items as unfrozen.If there are more unfreeze items, the adapter calls the GET_EXTERNAL_UNFREEZE_LIST service again, using the same beginDate as it did initially. The adapter then receives the next block of items and a new corresponding endDate, which becomes the retrieveDate for this next block of items.

  When there are no more unfreeze items, the adapter must save the endDate for the last block of items received, as the adapter must use this endDate as the beginDate the next time it calls the CHECK_PENDING_EXTERNAL_TASK and GET_EXTERNAL_UNFREEZE_LIST services.
- The beginDate and endDate are defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).

❖ **(RH08)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the list of unfrozen items to the adapter, an endDate for the block of items, and a Boolean flag (hasMore) indicating if there are more unfrozen items. If there is an error, URM sends an error message (ermStatusMessage).

❖ **(RH09)** The adapter checks the status response.

❖ **(RH10)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.

❖ **(RH11)** If there is no error, the adapter starts processing the unfrozen items, selecting the first item on the list.

❖ **(RH12)** The adapter tags the item as unfrozen in the repository. If the adapter had created a "hold" copy of the item in the repository, the adapter removes that copy since it is no longer needed. It would also perform an item update on the item to remove the "hold" copy location from the metadata of the item. For details on the update process, see Updating Items in URM (page 3-7).

❖ **(RH13)** The adapter checks to see if there are more items on the list.

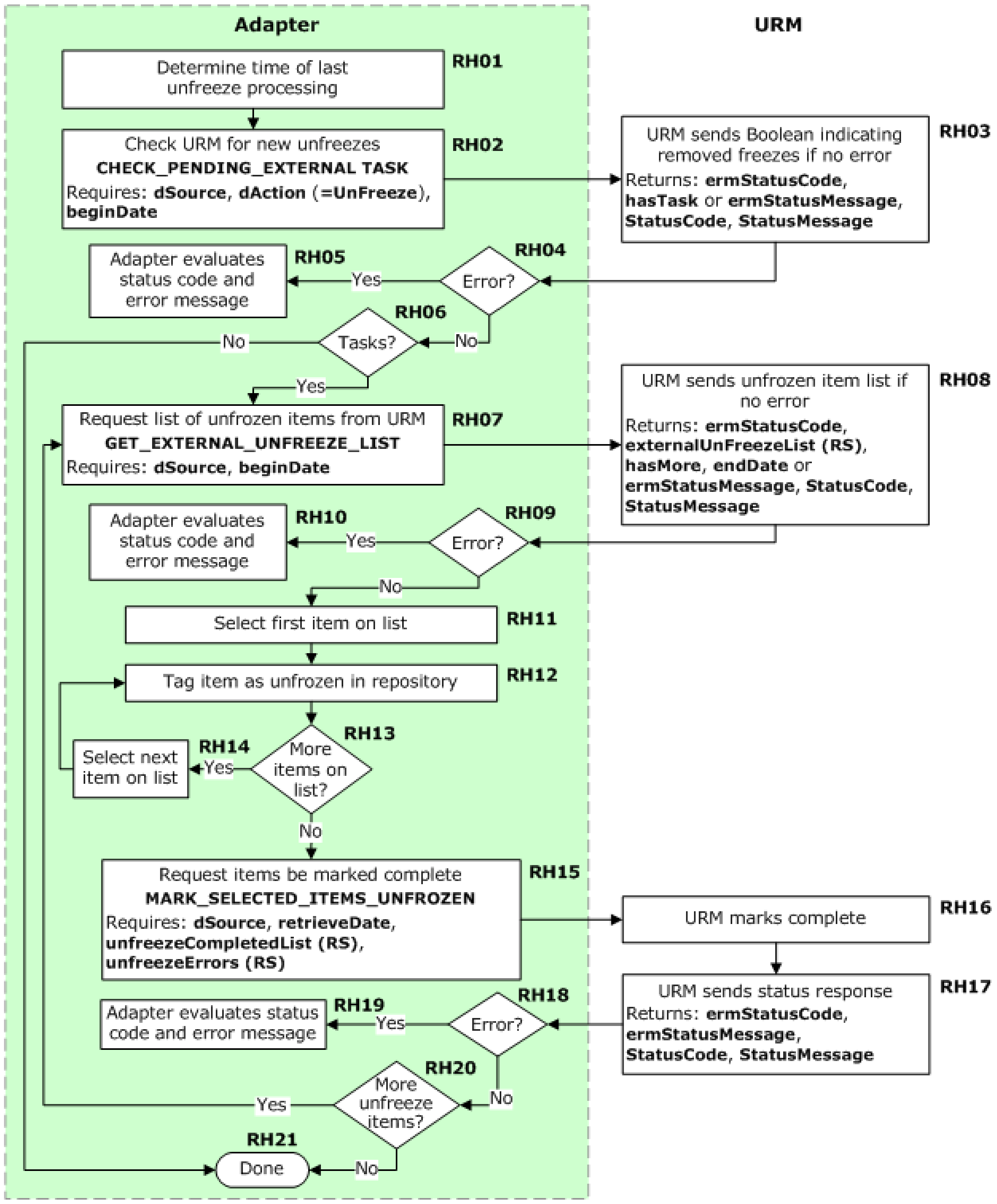
- **(RH14)** If there are more items on the list, the adapter selects the next item on the list and repeats the unfreeze process (RH11–RH13).
- **(RH15)** If there are no more items on the list, the adapter calls the MARK_SELECTED_ITEMS_UNFROZEN service to request URM to mark the unfreeze of the items as complete. For additional details, see MARK_SELECTED_ITEMS_UNFROZEN (page A-74).

Important: When calling the MARK_SELECTED_ITEMS_UNFROZEN service, you must include the retrieveDate parameter. It is critical that the retrieveDate match, exactly, the endDate that was returned by URM for the corresponding block of items when the GET_EXTERNAL_UNFREEZE_LIST service was called (RH07–RH08). The retrieveDate is defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).

- **(RH16)** URM marks the unfreeze of the items as complete.
- **(RH17)** URM sends a status response to the adapter. The status response either indicates that the unfreeze of the items has been marked as complete (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).
- **(RH18)** The adapter checks the status response.
- **(RH19)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(RH20)** If there is no error, the adapter checks to see if there are more unfreeze items (checks hasMore Boolean flag, returned in RH08). If there are more unfreeze items, the adapter requests the next set of items and repeats the unfreeze process (RH12–RH20).
- **(RH21)** If there are no more unfreeze items, the adapter stops unfreeze processing.

Note: When there are no more unfreeze items, the adapter must save the endDate for the last block of items received, as the adapter must use this endDate as the beginDate the next time it calls the CHECK_PENDING_EXTERNAL_TASK service (RH02) and the GET_EXTERNAL_UNFREEZE_LIST service (RH07).

Figure 4-3 Removing a hold/freeze


Adapter
URM
Determine time of last unfreeze processing
RH01
Check URM for new unfreezes
CHECK_PENDING_EXTERNAL TASK
Requires: dSource, dAction (=UnFreeze), beginDate
RH02
URM sends Boolean indicating removed freezes if no error
Returns: ermStatusCode, hasTask or ermStatusMessage, StatusCode, StatusMessage
RH03
Adapter evaluates status code and error message
RH05
Yes
Error?
RH04
RH06
No
Tasks?
No
Yes
Request list of unfrozen items from URM
GET_EXTERNAL_UNFREEZE_LIST
Requires: dSource, beginDate
RH07
URM sends unfrozen item list if no error
Returns: ermStatusCode, externalUnFreezeList (RS), hasMore, endDate or ermStatusMessage, StatusCode, StatusMessage
RH08
Adapter evaluates status code and error message
RH10
Yes
Error?
RH09
No
Select first item on list
RH11
Tag item as unfrozen in repository
RH12
RH13
Select next item on list
RH14
Yes
More items on list?
No
Request items be marked complete
MARK_SELECTED_ITEMS_UNFROZEN
Requires: dSource, retrieveDate, unfreezeCompletedList (RS), unfreezeErrors (RS)
RH15
URM marks complete
RH16
URM sends status response
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
RH17
Adapter evaluates status code and error message
RH19
Yes
Error?
RH18
RH20
Yes
More unfreeze items?
No
RH21
Done
No

Chapter 5

QUERYING URM

OVERVIEW

This section covers the following topics:

REQUESTING A RETENTION SCHEDULE

Certain adapters might want to auto-categorize items and associate them with specific retention schedules within the file plan. To do so they would need to download a file plan. The setup can be performed by an administrative user who should be able to view the entire file plan. Alternately, it can be performed by a user who is limited by security and should only see certain sections of the file plan.

This section covers the following topics:

Downloading the Entire Retention Schedule

The entire retention schedule can be downloaded as a single XML block. The process for downloading the entire retention schedule is as follows (see Figure 5-1):

❖ **(DE01)** The adapter calls the GET_FILE_PLAN_ALL service to request the entire file plan. For additional details, see GET_FILE_PLAN_ALL (page A-57).

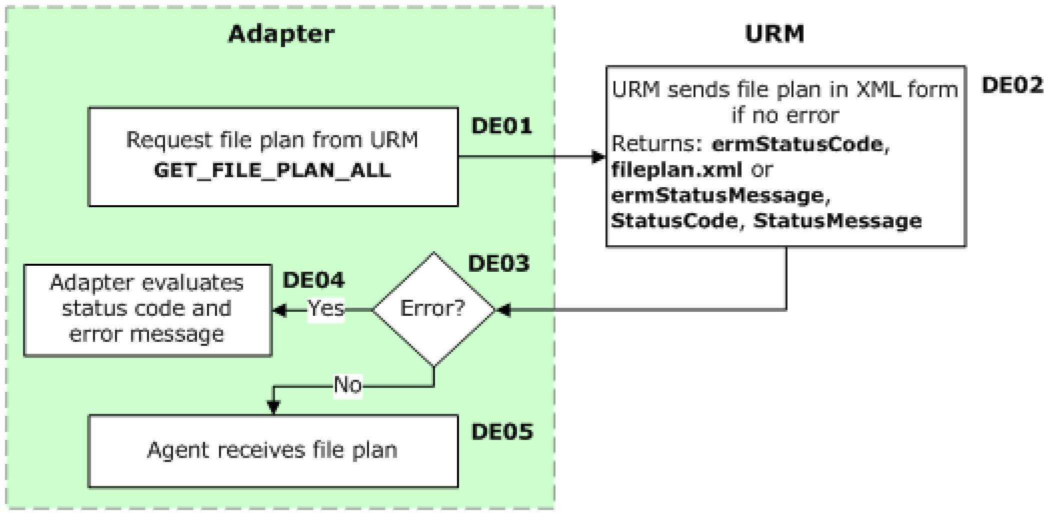
❖ **(DE02)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the entire file plan to the adapter as a single XML block. If there is an error, URM sends an error message (ermStatusMessage).

❖ **(DE03)** The adapter checks the status response.

❖ **(DE04)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.

❖ **(DE05)** If there is no error, the adapter receives the entire file plan.

Figure 5-1 Downloading the entire retention schedule

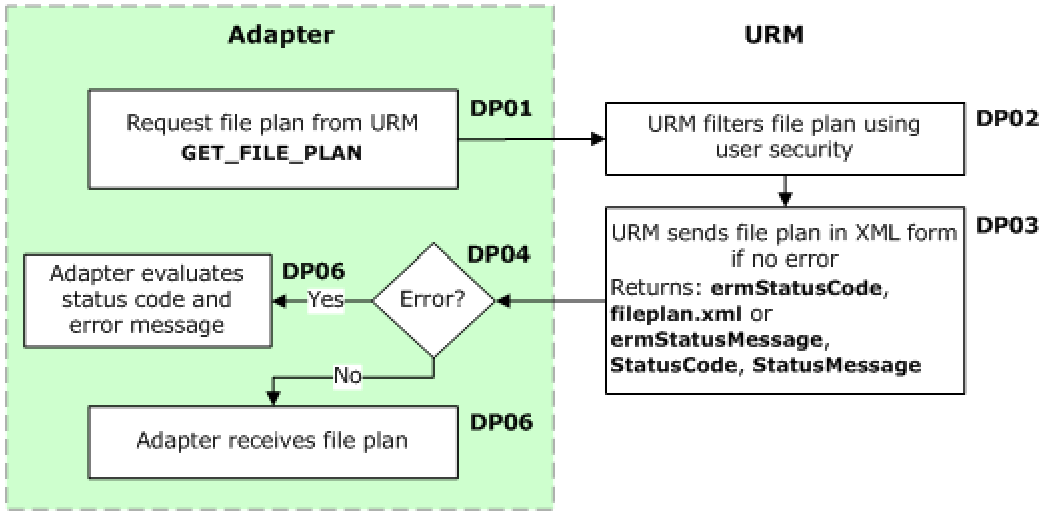

Adapter
URM
Request file plan from URM
GET_FILE_PLAN_ALL
DE01
URM sends file plan in XML form if no error
Returns: ermStatusCode, fileplan.xml or ermStatusMessage, StatusCode, StatusMessage
DE02
DE03
Error?
Yes
DE04
Adapter evaluates status code and error message
No
Agent receives file plan
DE05

Downloading Parts of the Retention Schedule for Viewing

Only aspects of the retention schedule that a particular user can view can be downloaded. The process for downloading parts of retention schedule is as follows (see Figure 5-2):

- **(DP01)** The adapter calls the GET_FILE_PLAN service to request the file plan. For additional details, see GET_FILE_PLAN (page A-56).
- **(DP02)** URM filters the file plan using the user's security.
- **(DP03)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the file plan to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- **(DP04)** The adapter checks the status response.
- **(DP05)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(DP06)** If there is no error, the adapter receives the file plan.

Figure 5-2 Downloading parts of the retention schedule

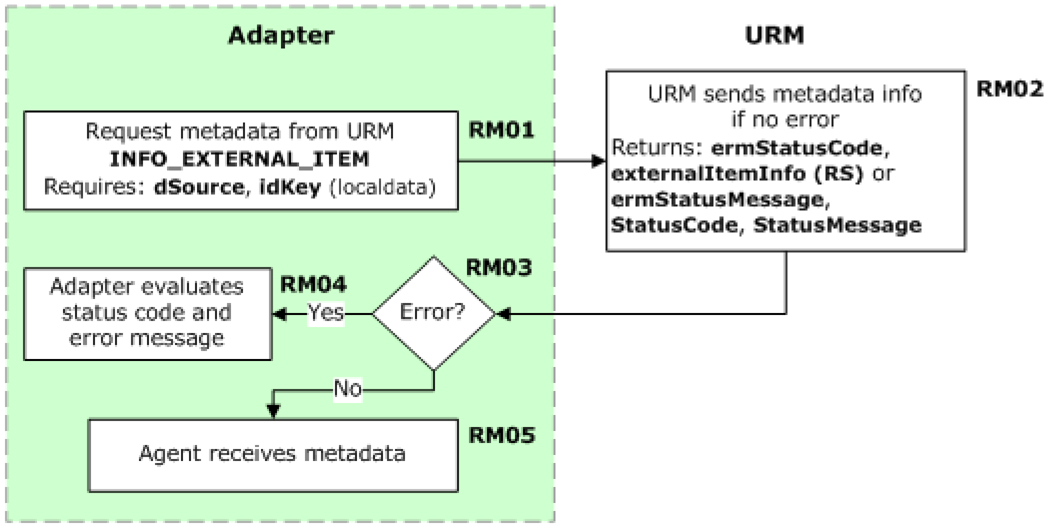

Adapter
URM
Request file plan from URM GET_FILE_PLAN
DP01
URM filters file plan using user security
DP02
URM sends file plan in XML form if no error
Returns: ermStatusCode, fileplan.xml or ermStatusMessage, StatusCode, StatusMessage
DP03
DP04
Error?
Yes
DP06
Adapter evaluates status code and error message
No
Adapter receives file plan
DP06

Requesting URM Metadata for an Item

The process for requesting URM metadata for a specific item is as follows (see Figure 5-3):

- **(RM01)** The adapter calls the INFO_EXTERNAL_ITEM service to request metadata for the item from URM. For additional details, see INFO_EXTERNAL_ITEM (page A-64).
- **(RM02)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the metadata to the adapter. If there is an error, URM sends an error message (ermStatusMessage).
- **(RM03)** The adapter checks the status response.
- **(RM04)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(RM05)** If there is no error, the adapter receives the metadata.

Figure 5-3 Requesting item metadata


Adapter
URM
Request metadata from URM
INFO_EXTERNAL_ITEM
Requires: dSource, idKey (localdata)
RM01
URM sends metadata info if no error
Returns: ermStatusCode, externalItemInfo (RS) or ermStatusMessage, StatusCode, StatusMessage
RM02
RM03
Error?
Yes
No
Adapter evaluates status code and error message
RM04
Agent receives metadata
RM05

Requesting the Lifecycle for an Item

The process for requesting the lifecycle for a specific item is as follows (see Figure 5-4):

❖ **(RL01)** The adapter calls the GET_LIFECYCLE_FOR_EXTERNAL_ITEM service to request the lifecycle for an item from URM. For additional details, see GET_LIFECYCLE_FOR_EXTERNAL_ITEM (page A-58).

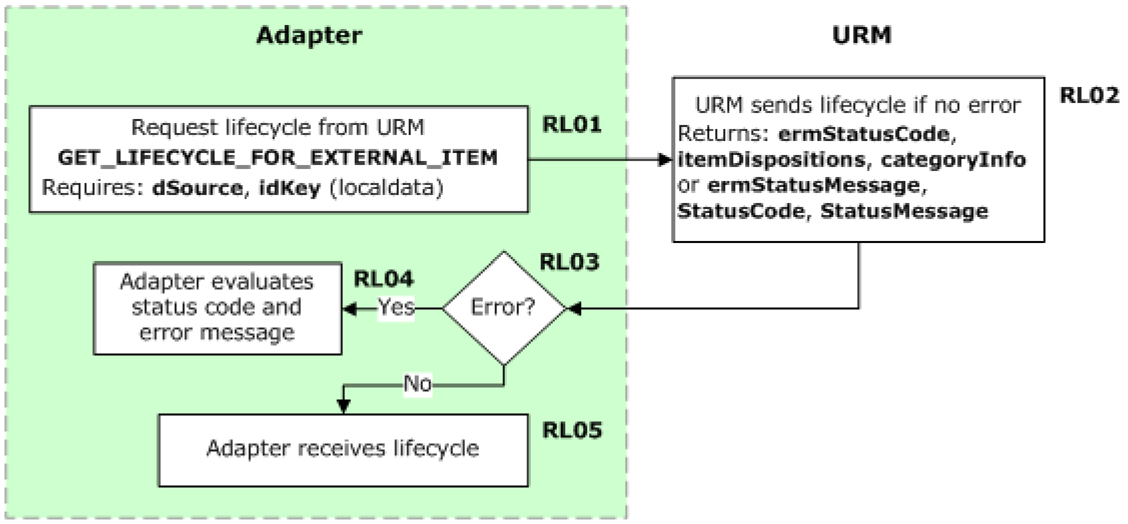
❖ **(RL02)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends the lifecycle to the adapter. If there is an error, URM sends an error message (ermStatusMessage).

❖ **(RL03)** The adapter checks the status response.

❖ **(RL04)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.

❖ **(RL05)** If there is no error, the adapter receives the lifecycle.

Figure 5-4 Requesting the lifecycle for an item


Adapter
URM
Request lifecycle from URM
GET_LIFECYCLE_FOR_EXTERNAL_ITEM
Requires: dSource, idKey (localdata)
RL01
URM sends lifecycle if no error
Returns: ermStatusCode, itemDispositions, categoryInfo or ermStatusMessage, StatusCode, StatusMessage
RL02
RL03
Error?
RL04
Yes
Adapter evaluates status code and error message
No
Adapter receives lifecycle
RL05

Chapter

6

MANAGING COMMUNICATIONS

OVERVIEW

This section covers the following topics:

Checking the Status of Individual Batch Tasks

When the adapter submits a batch task to URM, URM sends a task id that the adapter can use to monitor the task and determine whether or not the batch submission was processed successfully. The process for checking the status of an individual batch task is as follows (see Figure 6-1):

Note: The adapter can also check the status of multiple batch tasks. However, the adapter must check the status of an individual batch task to obtain a listing of any errors for that task. For details, see Checking the Status of Multiple Batch Tasks (page 6-5).

- **(CS01)** The adapter calls the CHECK_TASK_STATUS service to request the status of the individual batch task from URM. For additional details, see CHECK_TASK_STATUS (page A-14).
- **(CS02)** URM sends a status response to the adapter. If there is no error executing the service call (ermStatusCode=0), URM sends a Boolean flag (isTaskCompleted) indicating whether or not the task is completed. If the task is completed, URM also sends a Boolean flag (hasError) indicating whether or not there were errors with the task. If there is an error executing the service call, URM sends an error message (ermStatusMessage).
- **(CS03)** The adapter checks the status response.
- **(CS04)** If there is an error executing the service call, the adapter evaluates the status code and error message and determines how to proceed.
- **(CS05)** If there is no error executing the service call, the adapter checks to see if the task is complete.
- **(CS06)** If the task is not complete (isTaskCompleted=false), the adapter waits for a pre-configured period and then repeats the request for the batch task status (MS01).
- **(CS07)** If the task is complete (isTaskCompleted=true), the adapter checks to see if there were any errors with the task. If hasError is true, the error is either with the batch (a batchErrorMessage is present), or there is an error with one or more items within the batch (which will be enumerated in the errorList result set).

  If there is an error with the batch, isTaskCompleted will be true, hasError will be true, and a batchErrorMessage will be present. In this case, the entire batch has failed and has been removed from the scheduled queue. The batchErrorMessage will describe what caused the batch failure. The problem will need to be corrected, and the batch will need to be resubmitted. If there is an error with the batch, no items will have been processed, and thus the returned errorList will not be of value.

If the batch is processed but there is an error with one or more items within the batch, isTaskCompleted and hasError will be true, however a batchErrorMessage will not be present. The returned errorList will list the items in the batch that had errors and the types of the errors. This result set contains three fields:

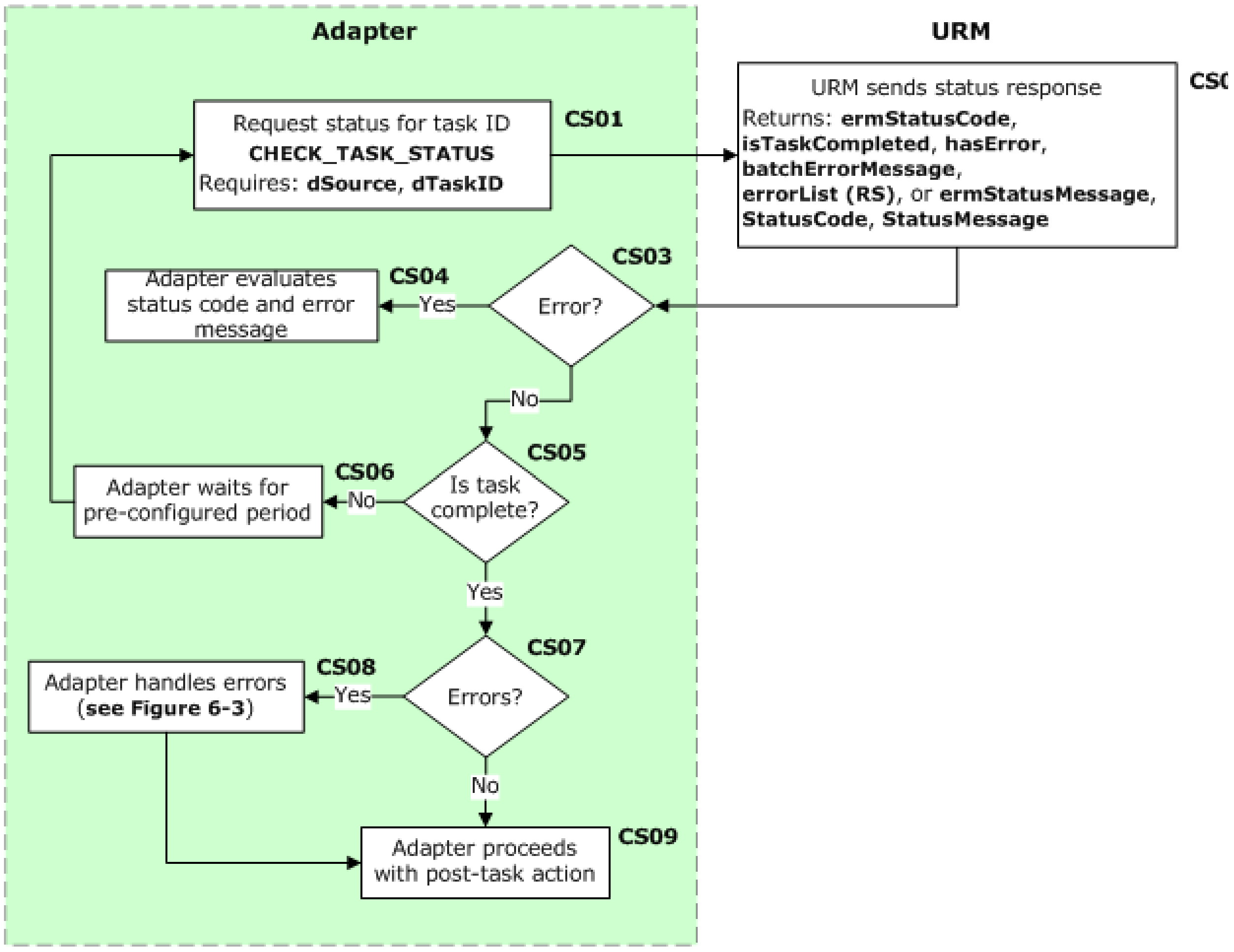
- **idKey**—the adapter's key (String).
- **errorCode**—the error code (Int).
- **errorMessage**—the error message (String).

Any items in the batch that are not in the errorList were processed successfully.

❖ **(CS08)** If there were errors with one or more items within the batch, the adapter handles the errors before proceeding with the post-task action (CS09). For more information, see Handling Task Status Errors (page 6-7).

❖ **(CS09)** If there were no errors with the task (hasError=false), The adapter proceeds with the post-task action.

Figure 6-1 Checking the status of an individual batch task


Adapter
URM
Request status for task ID
CHECK_TASK_STATUS
Requires: dSource, dTaskID
CS01
URM sends status response
Returns: ermStatusCode, isTaskCompleted, hasError, batchErrorMessage, errorList (RS), or ermStatusMessage, StatusCode, StatusMessage
CS0
CS03
Error?
Yes
Adapter evaluates status code and error message
CS04
No
CS05
Is task complete?
No
Adapter waits for pre-configured period
CS06
Yes
CS07
Errors?
Yes
Adapter handles errors (see Figure 6-3)
CS08
No
Adapter proceeds with post-task action
CS09

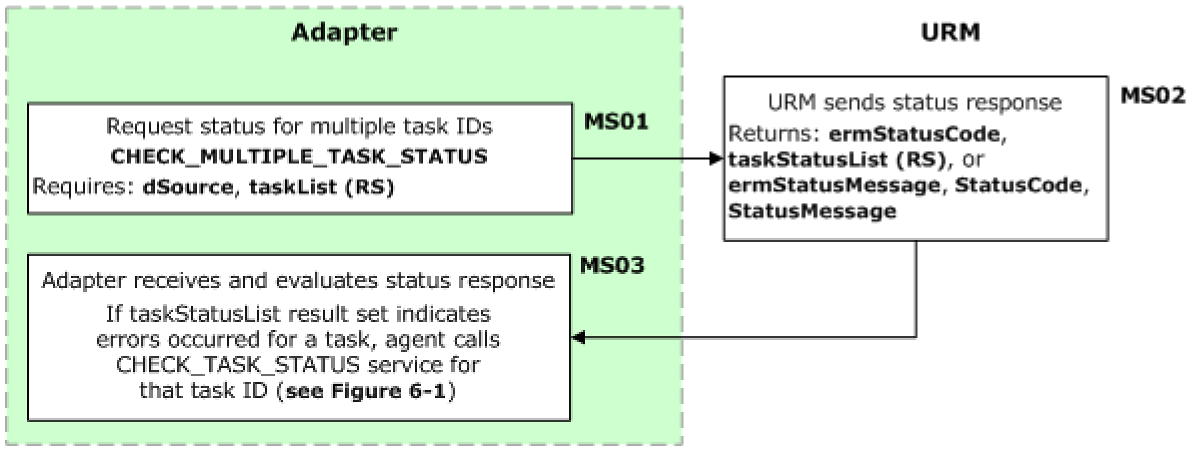
Checking the Status of Multiple Batch Tasks

When the adapter submits a batch task to URM, URM sends a task id that the adapter can use to monitor the task and determine whether or not the batch submission was processed successfully. The process for checking the status of multiple batch tasks is as follows (see Figure 6-2):

Note: The adapter must check the status of an individual batch task to obtain a listing of any errors for that task. For details, see Checking the Status of Individual Batch Tasks (page 6-2).

- **(MS01)** The adapter calls the CHECK_MULTIPLE_TASK_STATUS service to request the status of the batch tasks from URM. The adapter sends a result set listing the task IDs to check. For additional details, see CHECK_MULTIPLE_TASK_STATUS (page A-9).
- **(MS02)** URM sends a status response to the adapter. If there is no error executing the service call (ermStatusCode=0), URM sends a result set listing three pieces of information for each task: if the task is a valid scheduled task, if the task is complete, and if the task had any errors. If there is an error executing the service call, URM sends an error message (ermStatusMessage).
- **(MS03)** The adapter checks the status response, and it loops through the taskStatusList result set (if returned). For each task, the adapter checks to see if the task is valid, if it is complete, and if there were any errors. If the task is complete but had errors, the adapter must call the CHECK_TASK_STATUS service for that task ID to obtain a listing of the errors. For details, see Checking the Status of Individual Batch Tasks (page 6-2).

Figure 6-2 Checking the status of multiple batch tasks

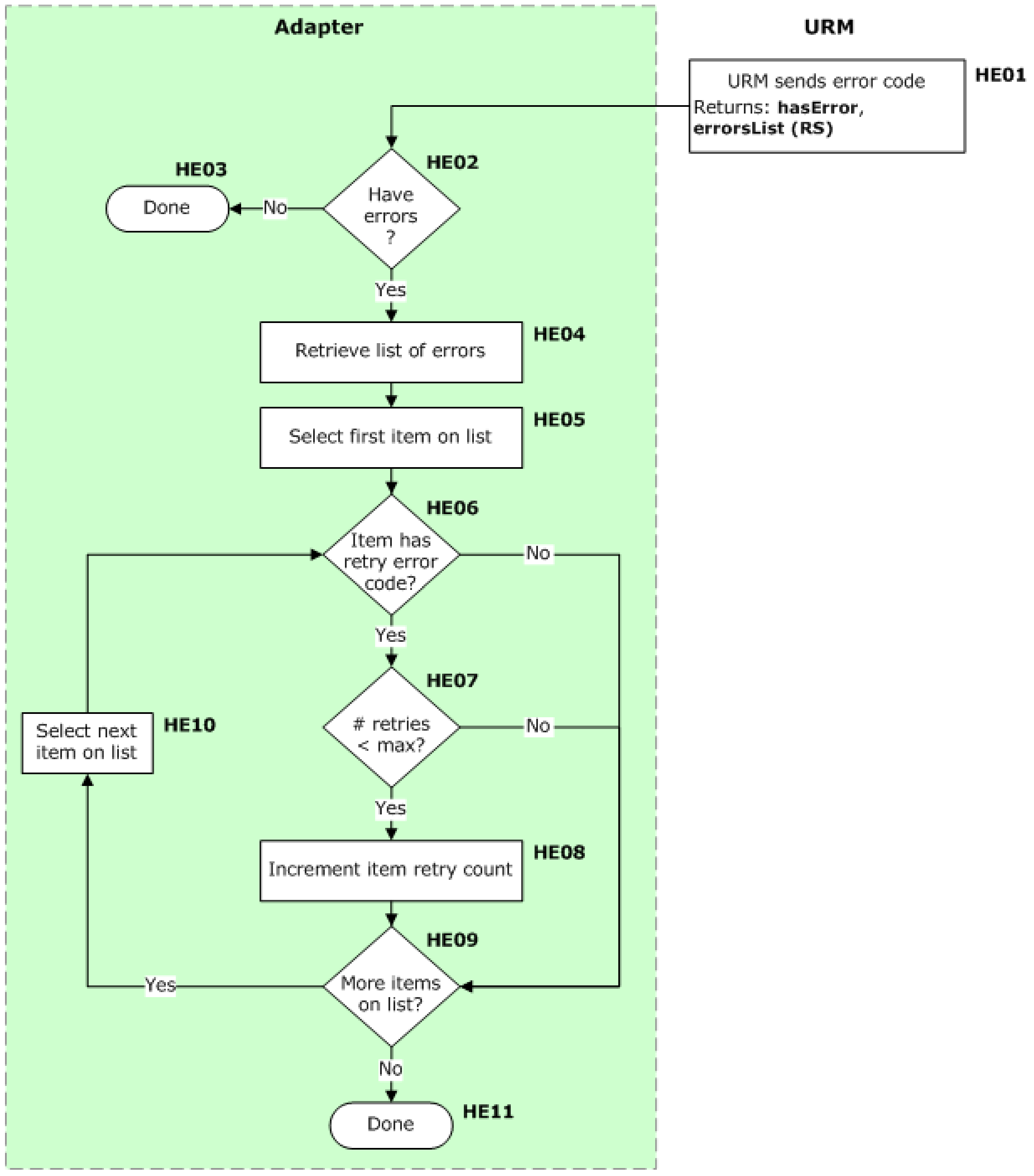

Adapter
URM
Request status for multiple task IDs
CHECK_MULTIPLE_TASK_STATUS
Requires: **dSource**, **taskList (RS)**
MS01
URM sends status response
Returns: **ermStatusCode**, **taskStatusList (RS)**, or **ermStatusMessage**, **StatusCode**, **StatusMessage**
MS02
Adapter receives and evaluates status response
If taskStatusList result set indicates errors occurred for a task, agent calls CHECK_TASK_STATUS service for that task ID (**see Figure 6-1**)
MS03

HANDLING TASK STATUS ERRORS

There might be errors when a batch task is processed by URM. In this case, URM sends an errorList result set to the adapter. The adapter must handle the errors before proceeding with the post-task action. The following is an example process that the adapter could use to handle the task status errors (see Figure 6-3):

- **(HE01)** URM sends an error code to the adapter. If there were errors, URM also sends an error list.
- **(HE02)** The adapter checks the error code.
- **(HE03)** If there were no errors, the process is complete.
- **(HE04)** If there were errors, the adapter retrieves a list of errors from the response.
- **(HE05)** The adapter selects the first item on the list.
- **(HE06)** The adapter checks to see if the item includes a retry error code. If the item does not include a retry error code, the adapter stops retrying that item and checks to see if there are more items on the list of errors (HE09).
- **(HE07)** If the item has a retry error code, the adapter checks to see whether the number of retries for that item is less than the maximum number of retries. If the number of retries for that item is not less than the maximum number of retries, the adapter stops retrying that item and checks to see if there are more items on the list of errors (HE09).
- **(HE08)** If the number of retries for that item is less than the maximum number of retries, the adapter increments the item retry count by one.
- **(HE09)** The adapter checks to see if there are more items on the list of errors.
- **(HE10)** If there are more items on the list of errors, the adapter selects the next item on the list and processes that item (HE06–HE09).
- **(HE11)** If there are no more items, the process is complete.

Figure 6-3 Handling task status errors



Segmenting Response Data

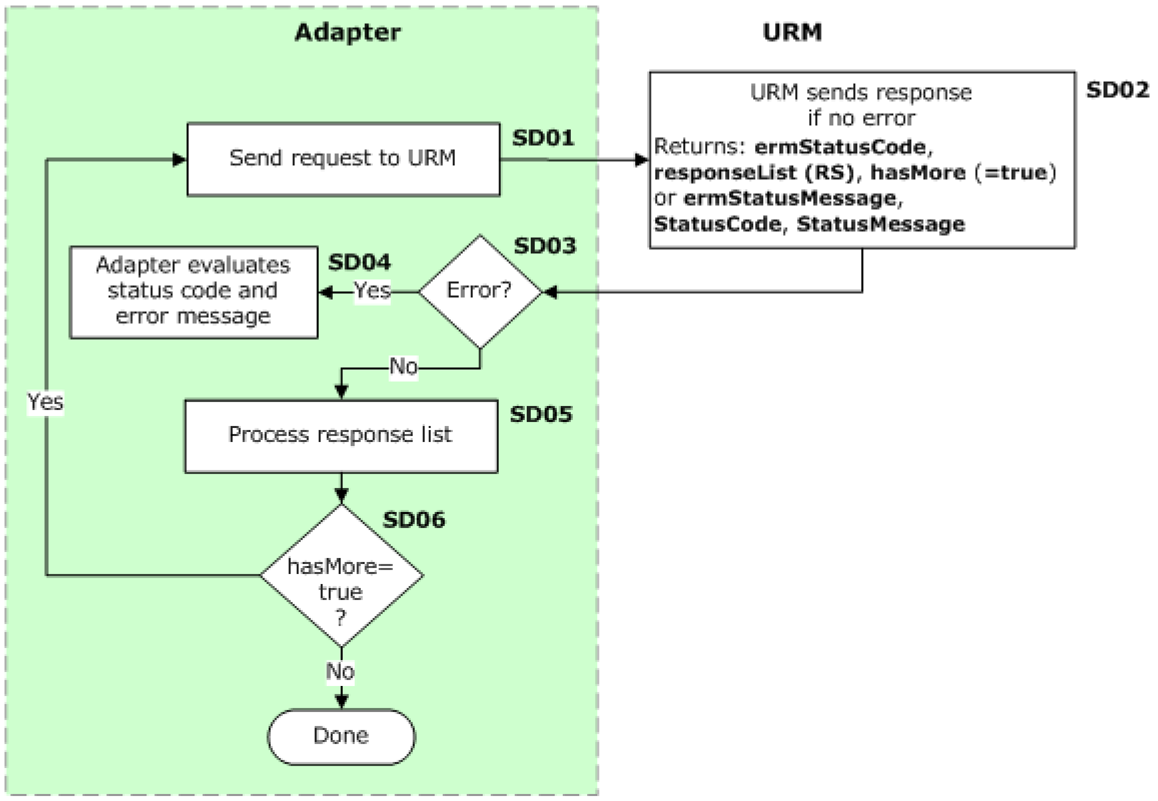
The adapter often needs to retrieve a list of items for URM that might be too long to transmit in one request. For example, the list of items to dispose of or the list of items to freeze might be too large to send in a single response. When that happens, URM can return only a portion, or chunk, of the items in the list and tell the adapter to ask for more items after processing that first chunk.

The process for segmenting response data is as follows (see Figure 6-4):

- **(SD01)** The adapter sends a request to URM.
- **(SD02)** URM sends a status response to the adapter. If there is no error (ermStatusCode=0), URM sends a partial response list to the adapter, along with a flag indicating that there are more items in the list that the adapter should retrieve in a subsequent request. If there is an error, URM sends an error message (ermStatusMessage).
- **(SD03)** The adapter checks the status response.
- **(SD04)** If there is an error, the adapter evaluates the status code and error message and determines how to proceed.
- **(SD05)** If there is no error, the adapter processes the response list.
- **(SD06)** The adapter checks to see if the flag indicates that there are more items to process. If the flag indicates that there are more items, the adapter repeats the process by sending another request (SD01). If the flag does not indicate that there are more items to process, the process is complete.

Tech Tip: It is recommended that the adapter also segment large listings that it sends to the server.

Figure 6-4 Segmenting response data


Adapter
URM
Send request to URM
SD01
URM sends response if no error
Returns: **ermStatusCode**, **responseList (RS)**, **hasMore** (**=true**) or **ermStatusMessage**, **StatusCode**, **StatusMessage**
SD02
Error?
SD03
Yes
Adapter evaluates status code and error message
SD04
No
Process response list
SD05
hasMore= true ?
SD06
Yes
No
Done

UPLOADING EXTERNAL LOG FILES

The adapter can upload log files to URM. The process for uploading log files is as follows (see Figure 6-5):

- **(UL01)** The adapter calls the UPLOAD_EXTERNAL_LOG_FILE service to upload the log files to URM.

  The adapter must include the log ID and a zip file containing the log files and an *index.hda* file. For additional details, see UPLOAD_EXTERNAL_LOG_FILE (page A-96).

- **(UL02)** URM uploads the log files.

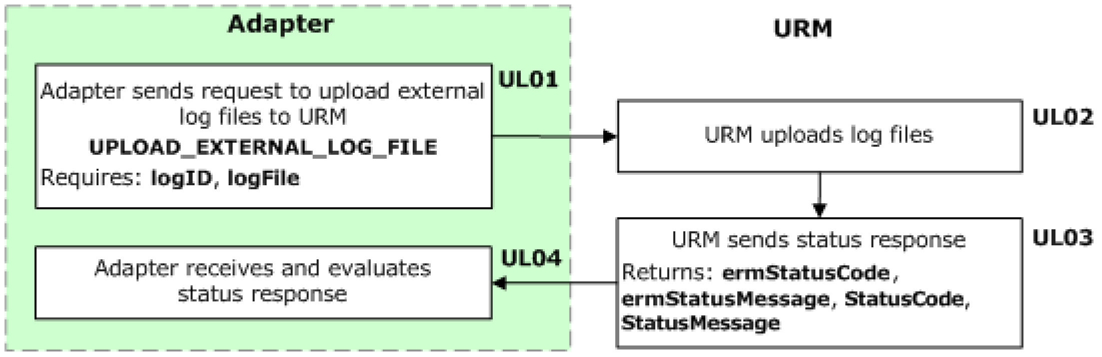
  By default, URM will store a maximum of 30 external log files. URM will upload up to 30 external log files, if they are newer than any files already stored and if they do not already exist. You can change the maximum number of external log files that URM will store by adding the `MaxNumberOfExternalLogFiles` configuration variable to the *config.cfg* file for the URM Server.

- **(UL03)** URM sends a status response. The status response either indicates that the upload was successful (ermStatusCode=0), or that there was an error. If there was an error, URM also sends an error message (ermStatusMessage).

- **(UL04)** The adapter receives and evaluates the status response.

Note: You can view external log files in the URM interface by navigating to Administration—Log Files—URM Logs. Links to all external log files stored in URM are displayed (by default, this is a maximum of 30 external log files).

Figure 6-5 Uploading an external log file

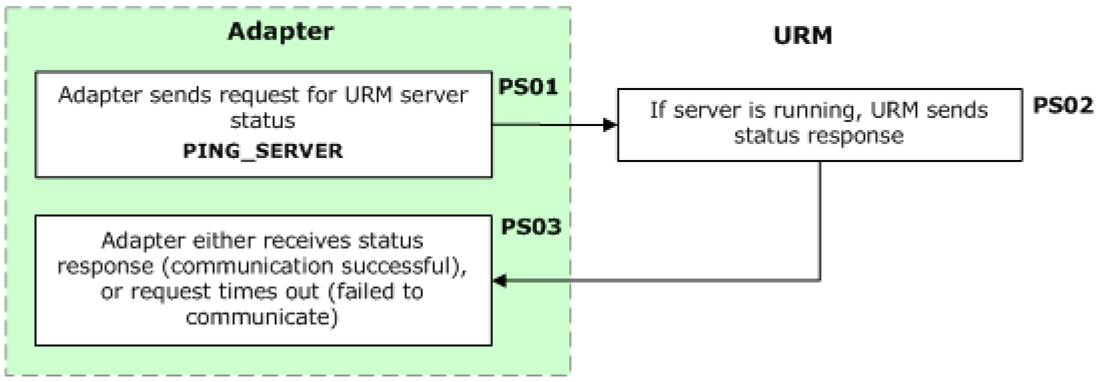

Adapter
URM
Adapter sends request to upload external log files to URM
UPLOAD_EXTERNAL_LOG_FILE
Requires: logID, logFile
UL01
URM uploads log files
UL02
URM sends status response
Returns: ermStatusCode, ermStatusMessage, StatusCode, StatusMessage
UL03
Adapter receives and evaluates status response
UL04

PINGING THE URM SERVER

The adapter can ping the URM server to see if it is running. The process for pinging the URM server is as follows (see Figure 6-6):

- **(PS01)** The adapter calls the PING_SERVER service. No parameters are required for this service. For additional details, see PING_SERVER (page A-76).
- **(PS02)** If the URM server is running, URM sends a status response.
- **(PS03)** The adapter either receives a status response (indicating that the communication with the URM server was successful), or the request times out (indicating that the adapter failed to communicate with the URM server).

Figure 6-6 Pinging the URM server


Adapter
URM
Adapter sends request for URM server status
PING_SERVER
PS01
If server is running, URM sends status response
PS02
PS03
Adapter either receives status response (communication successful), or request times out (failed to communicate)

Appendix

A

URM Adapter Services

Overview

This appendix contains a list of the main services that are used for communications between URM and adapters:

About Error Codes

URM error codes (`ermStatusCode`, which is returned by every service, and `errorCode`, which is returned in the `errorList` result set for each item in a batch task that had an error) fall into the following three categories:

Error Code	Description
100–199	Retry error codes.
200–299	Do not retry error codes.
300–399	Setup error codes.

StatusCode and StatusMessage

StatusCode and StatusMessage are secondary. If ermStatusCode is not returned, there is a low-level Content Server problem, and the adapter should look at the StatusCode and StatusMessage that is returned.

About the WSDL Generator Component

A WSDL Generator component is provided with URM. The WSDL Generator component allows for creating WSDLs for the services that URM adapters use to communicate with the URM Server. Users can then take the WSDLs and plug them into APIs to create web services that can be used with the URM Server.

The WsdlGenerator component (*WsdlGenerator.zip*), the Wsdl function call generator (*WsdlDetails.zip*), and samples of custom WSDLs are provided on the URM media. After installation, they are stored in the *<install_dir>/custom* directory.

Tech Tip: When URM is installed, a custom WSDL file (*wsdl_custom.hda*) that includes the URM API is merged with any custom WSDL files already on the content server. The URM installer attempts to place the merged *wsdl_custom.hda* file in the *<install_dir>/data/soap/custom* directory. The URM *wsdl_custom.hda* file is available with the URM component files in the *<install_dir>/custom/ExternalHelper/data/soap/custom* directory.

The example WSDL services provided in this appendix for each service were created using the WSDL Generator component. For more information about using the WSDL Generator component, refer to the *Using WSDL Generator and SOAP* guide that is included in the *WsdlGenerator.zip* file.

SERVICES

This section covers the following URM adapter services:

ADD_EXTERNAL_CUSTOM_FIELDS

This service is used to add external custom fields after registering the adapter. This service is used in the following process:

❖ Adding External Custom Fields (page 2-9)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
externalCustomMetaDefinition	A content server result set. Specify any adapter-specific metadata fields. Please note the following important considerations: • If a field already exists, you will receive an error. • **dOptionListType**—only the following values are valid: combo, multi, or strict • **dType—**only the following values are valid: Date, Int, Text, BigText, or Memo • **dIsDisplayOnly—**indicates if the field is a placeholder field. If the field is defined as a placeholder field, URM does not create the field in the source table. • If you want to maintain the original field data when it needs to be truncated, consider creating a custom field, with the same name as the mapped field, that is long enough to hold the original data. For example, consider you have a Subject field that you map to dDocTitle, which is indicated as a truncated field. You could create a custom field named Subject. Then you only need to send the Subject field value once, as URM will truncate the data if it is greater than the length of the field for dDocTitle. The full value will still be in the Subject field.

Example externalCustomMetaDefinition Result Set

The following is an example externalCustomMetaDefinition result set in an .hda file format. Please note that this format is not required; other formats can be used.

```
<?hda version="7.5.1 (050330)" jcharset=UTF8 encoding=utf-8?>
@Properties LocalData
blFieldTypes=
blDateFormat=M/d/yy {h:mm[:ss] {aa}[zzz]}!mAM,PM!tAmerica/Chicago
@end
@ResultSet externalCustomMetadataDefinition
14
dName
dCaption
dType
dIsRequired
dIsEnabled
dIsSearchable
dIsOptionList
dDefaultValue
dOptionListKey
dOptionListType
dHides
dRequires
dOrder
dIsDisplayOnly
ExternalComments
External Comments
Memo
false
true
true
false

5
false
Department
Department
Text
true
true
true
false

6
false
Color
Color Option List Field
Text
false
```

```
true
true
true
Red
ColorList
strict

7
false
ExternalBigText
External Big Text Field
BigText
false
true
true
false

8
false
@end
```

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 301 – Setup error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
AddExternalCustomFields (String dSource, ExternalCustomMetaDefinition
externalCustomMetaDefinition, IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

ExternalCustomMetaDefinition (page A-100)
IdcProperty (page A-101)
StatusInfo (page A-101)

CHECK_MULTIPLE_TASK_STATUS

This service is used to request the status multiple batch tasks from URM. The adapter sends a result set to URM listing the task IDs. URM returns a result set that includes three pieces of information for each task: if the task is a valid scheduled task, if the task is complete, and if the task had any errors.

This service is used in the following process:

❖ Checking the Status of Multiple Batch Tasks (page 6-5)

Note: The adapter must check the status of an individual batch task to obtain a listing of any errors for that task. For details, see Checking the Status of Individual Batch Tasks (page 6-2) and CHECK_TASK_STATUS (page A-14).

Required Parameter

This parameter must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
taskList	A content server result set that lists the task IDs to check. This result set contains one field: • **dTaskID**—the task ID that URM sent when the adapter submitted the batch task to URM.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 102 – Retrieval error.
taskStatusList	A content server result set that lists the status of each task. This result set contains four fields: • **dTaskID**—the task ID (String). • **isValidTaskID**—if the task is a valid scheduled task (Boolean). If the task is not valid, isTaskCompleted and hasError will be meaningless, although they will be false. • **isTaskCompleted**—if the task is complete (Boolean). If the task has not completed, hasError will be meaningless, although it will be false. • **hasError**—if the task had any errors (Boolean). This field is only meaningful if the task is valid and complete. If there are errors for a task, this field only indicates that errors occurred; this service does not return a listing of errors. The adapter must check the status of the individual task (using the task ID that has errors) to obtain a listing of the errors for that task. For details, see Checking the Status of Individual Batch Tasks (page 6-2) and CHECK_TASK_STATUS (page A-14).
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
CheckMultipleTaskStatus (TaskList taskList, String dSource,
IdcProperty[] extraProps)
```

Returns

```
TaskListStatus taskStatusList
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`TaskList` (page A-101)
`IdcProperty` (page A-101)
`TaskListStatus` (page A-101)
`StatusInfo` (page A-101)

CHECK_PENDING_EXTERNAL_TASK

This service is used to check if there are any pending tasks to perform. Possible tasks are Search, Disposition, Freeze, and UnFreeze.

This service is used in the following processes:

❖ Performing Federated Searches (page 4-1)

❖ Performing Dispositions (page 4-5)

❖ Performing Holds/Freezes (page 4-10)

❖ Removing Holds/Freezes (page 4-14)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
dAction	Possible actions (tasks) are Search, Disposition, Freeze, or UnFreeze.

Parameter	Description
beginDate	The time when the adapter last performed the specific type of task (Search, Disposition, Freeze, or UnFreeze). Please note the following important considerations: • If it is the adapter’s first time completing that task, the adapter should leave the beginDate empty. • If the adapter has completed a Disposition before, it should have saved the endDate it received from URM when it called the LIST_EXTERNAL_APPROVED_DISP_ACTIONS service. The adapter should use this endDate as the beginDate. • If the adapter has completed a Freeze or Unfreeze before, it should have saved the endDate it received from URM the last time it called the GET_EXTERNAL_FREEZE_LIST or GET_EXTERNAL_UNFREEZE_LIST service. The adapter should use this endDate as the beginDate. • The beginDate and endDate are defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 102 – Retrieval error.
hasTask	false – No task. true – Has task.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.

Parameter	Description
StatusMessage	Content Server message.

WSDL Service

Function

```
CheckPendingExternalTask (String dSource, String dAction, String beginDate,
IdcProperty[] extraProps)
```

Returns

```
boolean hasTask
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

CHECK_TASK_STATUS

This service is used to request the status of an individual batch task from URM. If the task is not complete, URM sends an incomplete status to the adapter. If the adapter receives an incomplete status from URM, it waits for a pre-configured period and then repeats the request for the batch task status. If the task is complete, URM checks to see if there were any errors when processing the task. If there were no errors, URM sends a complete status to the adapter.

This service is used in the following process:

❖ Checking the Status of Individual Batch Tasks (page 6-2)

Note: The adapter can also check the status of multiple batch tasks. However, the adapter must check the status of an individual batch task to obtain a listing of any errors for that task. For details, see Checking the Status of Multiple Batch Tasks (page 6-5) and CHECK_MULTIPLE_TASK_STATUS (page A-9).

Required Parameter

This parameter must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
dTaskID	The task ID that URM sent when the adapter submitted the batch task to URM.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 102 – Retrieval error.
isTaskCompleted	false – Not completed. true – Completed.

Parameter	Description
hasError	false – No error. true – Has error(s). If true, the error is either with the batch (a batchErrorMessage is present), or there is an error with one or more items within the batch (which will be enumerated in the errorList result set).
batchErrorMessage	If there is an error with the batch, hasError will be true, isTaskCompleted will be true, and a batchErrorMessage will be present. In this case, the entire batch has failed and has been removed from the scheduled queue. The batchErrorMessage will describe what caused the batch failure. The problem will need to be corrected, and the batch will need to be resubmitted.
errorList	Content server result set that lists the items in the batch that had errors and the types of the errors. If there is an error with the batch (hasError is true and a batchErrorMessage is present) no items will have been processed, and the returned errorList will not be of value. This result set contains three fields: • **idKey**—the adapter’s key (String). • **errorCode**—the error code (Int). • **errorMessage**—the error message (String). Any items in the batch that are not in the errorList were processed successfully.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
CheckTaskStatus (String dTaskID, String dSource, IdcProperty[] extraProps)
```

Returns

```
boolean isTaskCompleted
boolean hasError
ErrorList errorList
int ermStatusCode
String ermStatusMessage
String batchErrorMessage
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`ErrorList` (page A-99)
`StatusInfo` (page A-101)

CHECKIN_EXTERNAL

This service is used to declare new items to URM individually. This service is used in the following process:

❖ Declaring Individual Items (page 3-5)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
propertyList (localdata)	The adapter's metadata field value pairs. Please note the following important considerations: • **xCategoryID** and **xFolderID**—a record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.

Parameter	Description
StatusMessage	Content Server message.

WSDL Service

Function

```
CheckInExternal (String dSource, IdcProperty[] checkInValues,
IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

CHECKIN_INTERNAL

When a repository (such as a file server), is not able to preserve a record or non-record item over its retention period or purge the record properly when it is time for disposal, this service is used to check the item into the URM repository, so that the record can be preserved and purged properly. This service is used in the following process:

- Checking Items into URM (Internal Checkin) (page 3-22)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.

Parameter	Description
propertyList (localdata)	The adapter's metadata field value pairs. Please note the following important considerations: • **dDocAuthor**—the dDocAuthor field is optional. If it is not passed, URM will use the authenticated user. The dDocAuthor field can be specified in the profile if you want to overwrite the value that is used for authentication. • **xRMProfileTrigger**—if you want to use a profile to set values for fields, the profile must be set up in URM and you must map the xRMProfileTrigger field. • **dDocType**—the dDocType field must be set using a profile. • **dDocName and dDocTitle**—the dDocName and dDocTitle fields are usually mapped fields that are handled outside of profiles. • **dSecurityGroup**—if the dSecurityGroup field is not mapped or passed, URM will set it to the default security group for the source or to Public. • **xCategoryID** and **xFolderID**—a record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error.
primaryFile	The file to be uploaded to URM.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.

Parameter	Description
docUrl	The URL to the item in URM.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
CheckInInternal (String dSource, IdcProperty[] checkInValues, IdcFile primaryFile
IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
String docUrl
StatusInfo statusInfo
```

Complex Types

IdcFile (page A-101)
IdcProperty (page A-101)
StatusInfo (page A-101)

CHECKIN_MULTIPLE_EXTERNAL

This service is used to declare new items to URM in a batch. This service is used in the following process:

❖ Declaring Items in Batch (page 3-2)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
externalRecordsMetadataList	A content server result set of propertyLists. Please note the following important considerations: • **xCategoryID** and **xFolderID**—a record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
dTaskID	The task ID. This task ID is used for checking the status of the batch checkin.
ermStatusMessage	Error message.

Parameter	Description
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
CheckInMultipleExternal (String dSource,
ExternalMetaDataList externalRecordsMetadataList, IdcProperty[] extraProps)
```

Returns

```
String dTaskID
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

ExternalMetaDataList (page A-100)
IdcProperty (page A-101)
StatusInfo (page A-101)

CHECKIN_OR_EDIT_EXTERNAL

This service is used when an item in the repository either needs to be managed externally by URM or have its metadata updated in URM, but the adapter does not know if the item exists already in URM. In this case, the adapter can provide URM with item metadata and allow URM to determine whether the item should be declared or updated. If the item does not exist URM performs a declaration (external checkin; the items are stored externally in the repository, not in URM), and if the item does exist (are being managed externally already) URM performs an update.

This service is used in the following processes:

❖ Declaring or Updating Individual Items (page 3-15)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
propertyList (localdata)	The adapter's metadata field value pairs. Please note the following important considerations: • **xCategoryID** and **xFolderID**—a record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error. • Neither dDocName nor dLongName can be updated.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.

Parameter	Description
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
CheckInOrEditExternal (String dSource, IdcProperty[] checkInValues,
IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

IdcProperty (page A-101)
StatusInfo (page A-101)

CHECKIN_OR_EDIT_MULTIPLE_EXTERNAL

This service is used when items in the repository either need to be managed externally by URM or have their metadata updated in URM, but the adapter does not know if the items exist already in URM. In this case, the adapter can provide URM with item metadata and allow URM to determine whether the items should be declared or updated. If the items do not exist URM performs a declaration (external checkin; the items are stored externally in the repository, not in URM), and if the items do exist (are being managed externally already) URM performs an update.

This service is used in the following processes:

❖ Declaring or Updating Items in Batch (page 3-12)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
externalRecordsMetadataList	A content server result set of propertyLists. Please note the following important considerations: • **xCategoryID** and **xFolderID**—a record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error. • Neither dDocName nor dLongName can be updated.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
dTaskID	The task ID. This task ID is used for checking the status of the batch checkin.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
CheckInOrEditMultipleExternal (String dSource,
ExternalMetaDataList externalRecordsMetadataList, IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
String dTaskID
StatusInfo statusInfo
```

Complex Types

`ExternalMetaDataList` (page A-100)
`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

CONVERT_TRANSFERRED_ITEM_TO_LINK

This service is used to update the external reference in URM with the new ID when an item being managed externally is transferred to the URM repository. This service is used in the following processes:

❖ Transferring Items to URM (page 3-25)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
idKey (localdata)	The adapter’s key field value for the original item.
newIdKey (localdata)	The adapter’s key field value for the new placeholder for the docUrl.

Optional Parameter

This optional parameter may be specified:

Parameter	Description
editValues	This optional propertyList parameter enables the adapter to specify any metadata values that should be changed when the item is transferred to the URM repository. The propertyList only needs to include the metadata fields that should be changed. Any fields that are not specified will retain their original value.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
ermStatusMessage	Error message.

Parameter	Description
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
ConvertTransferredItemToLink (String dSource, String idKey, String newIdKey,
IdcProperty[] editValues, IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

CREATE_EXTERNAL_SOURCE

This service is used to register an adapter. This service is used to create a table for the adapter with a table name that is specified. This service is used in the following process:

❖ Registering Adapters (page 2-2)

Required Parameters

These parameters must be specified:

Parameter	**Description**
dSource	The adapter identifier (ID). Please note the following important considerations: • dSource must be unique. • The maximum length for dSource is 30 characters. If dTable (optional parameter) is not specified, dSource is used for dTable. Because the maximum length for dTable is 16 characters, in this case dSource must not exceed 16 characters or dTable will fail its maximum size requirement.

Parameter	Description
externalFieldMap	A content server result set with the columns rmField, externalField, and externalCaption. Specify the fields mapping between URM fields and the adapter's fields. Please note the following important considerations: • The adapter's unique ID field must be mapped to URM's dDocName or dLongName field. By default, dDocName is 100 characters long. If an adapter needs a long ID field, the optional keysize parameter must be specified. If the optional keysize parameter is specified, dLongName must be used. • By default, dDocTitle is 200 characters long. • If you want to maintain the original field data when it needs to be truncated, consider creating a custom field, with the same name as the mapped field, that is long enough to hold the original data. For example, consider you have a Subject field that you map to dDocTitle, which is indicated as a truncated field. You could create a custom field named Subject. Then you only need to send the Subject field value once, as URM will truncate the data if it is greater than the length of the field for dDocTitle. The full value will still be in the Subject field.

Example externalFieldMap Result Set

The following is an example externalFieldMap result set in an .hda file format. Please note that this format is not required; other formats can be used.

```
<?hda version="7.5.1 (050330)" jcharset=UTF8 encoding=utf-8?>
@Properties LocalData
blFieldTypes=
blDateFormat=M/d/yy {h:mm[:ss] {aa}[zzz]}!mAM,PM!tAmerica/Chicago
@end
@ResultSet externalFieldMap
3
rmField
externalField
externalCaption
dDocName
exDocName
EX Content ID
dSecurityGroup
exSecurityGroup
EX Security Group
dDocAccount
exDocAccount
EX Account
dDocTitle
exDocTitle
EX Title
dCreateDate
exCreateDate
EX Create Date
dDocAuthor
exAuthor
EX Author
xExternalLocation
exDocLocation
EX Doc Location
dLongName
exLongName
EX Long Name
dDocType
exDocType
EX Type
xIsRecord
exIsRecord
EX Is Record
xCategoryID
exCategoryID
EX Category
xFolderID
exFolderID
```

```
EX Folder
@end
```

Optional Parameters

These optional parameters may be specified:

Parameter	**Description**
dTable	A table name is optional, but specifying a table name is recommended. Please note the following important considerations: • If a table name is not specified, the source name (required dSource parameter) will be used to create a table for the adapter. • The maximum length for dTable is 16 characters. If dTable is not specified, dSource is used for dTable. Because the maximum length for dTable is 16 characters, in this case dSource must not exceed 16 characters or dTable will fail its maximum size requirement.

Parameter	Description
externalCustomMetaDefinition	A content server result set. Specify any adapter-specific metadata fields. Please note the following important considerations: • **dOptionListType**—only the following values are valid: combo, multi, or strict • **dType—**only the following values are valid: Date, Int, Text, BigText, or Memo • **dIsDisplayOnly—**indicates if the field is a placeholder field. If the field is defined as a placeholder field, URM does not create the field in the source table. • **dLength—**specifies the length of the field. The following are the length ranges and default lengths for custom fields (if the length is not specified, URM will use the default length for each field type): • Text field: 1–100; default 100 • BigText: 101–200; default 200 • Memo field: 201–2Gb (MS SQL), 201–4000 (Oracle), 201–2000 (Oracle Japanese); default 1000 for all • If you are using Microsoft SQL Server 2005, you should set the EnableLongMemoFieldForSCS7 configuration variable in the content server instance where URM is installed. When the EnableLongMemoFieldForSCS7 configuration variable is set to true, during table creation the length of the Memo type is 2,000,000,000. When this configuration variable is set to false, the length of the Memo type is 1000 (note that is specific to URM; the default Memo type length for Content Server is 255). It is recommended that you set Content Server configuration variables using Admin Server. Content Server configuration variables can also be set by editing the *config.cfg* file located in the *<install_dir>/config* directory. **Note:** When using any database other than Microsoft SQL Server 2005, the EnableLongMemoFieldForSCS7 configuration variable has no effect on content server functionality.

Parameter	Description
blocksize	The adapter can specify the blocksize to control the number of rows in the response result set.
keysize	If the adapter needs to have a long ID, the keysize must be specified. Please note the following important considerations: • By default, the keysize must be greater than 100 and less than or equal to 1,000. • You can change the maximum keysize value using ExternalMaxKeySize in URM, however this is not recommended. • If you specify a keysize, you must map dLongName in externalFieldMap.
sourceDisplayName	If sourceDisplayName is not specified, dSource will be used for the display name. Specify a sourceDisplayname if you want to use a different display name or you want to use spaces in the display name (dSource and dTable cannot contain spaces).

Example externalCustomMetaDefinition Result Set

The following is an example externalCustomMetaDefinition result set in an .hda file format. Please note that this format is not required; other formats can be used.

```
<?hda version="7.5.1 (050330)" jcharset=UTF8 encoding=utf-8?>
@Properties LocalData
blFieldTypes=
blDateFormat=M/d/yy {h:mm[:ss] {aa}[zzz]}!mAM,PM!tAmerica/Chicago
@end
@ResultSet externalCustomMetadataDefinition
14
dName
dCaption
dType
dIsRequired
dIsEnabled
dIsSearchable
dIsOptionList
dDefaultValue
dOptionListKey
dOptionListType
dHides
dRequires
```

```
dOrder
dIsDisplayOnly
ExternalComments
External Comments
Memo
false
true
true
false

5
false
Department
Department
Text
true
true
true
false

6
false
Color
Color Option List Field
Text
false
true
true
true
Red
ColorList
strict

7
false
ExternalBigText
External Big Text Field
BigText
false
true
true
false

8
false
@end
```

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 301 – Setup error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
CreateExternalSource (String dSource, String dTable,
ExternalFieldMap externalFieldMap,
ExternalCustomMetaDefinition externalCustomMetaDefinition, int blocksize,
int keysize, String sourceDisplayName, IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

ExternalFieldMap (page A-100)
ExternalCustomMetaDefinition (page A-100)
IdcProperty (page A-101)
StatusInfo (page A-101)

DELETE_BY_PATTERN

This service is used to delete items that match a given pattern. It deletes all items where the specified field starts with the specified pattern (assuming the items are not frozen). If the optional dateField and dateValue are specified, all items that match the pattern criteria and are older than the date specified for the dateField will be deleted. Items that match the pattern criteria but are newer than the date specified will not be deleted.

This service is used in the following processes:

- ❖ Deleting Items by Pattern (page 3-20)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
field	The name of the field to apply the pattern to. **Note:** Optionally, you can specify an operator for this parameter. If you do not specify an operator, a beginsWith will always be performed. For details, see Optional Parameters.
pattern	The pattern string to match against the specified field.

Optional Parameters

These optional parameters may be specified:

Parameter	Description
operator	The operator for the field parameter. The following operators are supported: • beginsWith (default) • endsWith • contains • equals The operator parameter is optional, and when it is not passed a beginsWith will always be performed. **Note:** The operator parameter applies to the field parameter. The dateField and dateValue parameters do not have an operator.
dateField	The name of the date field.
dateValue	The date you want to use when matching the pattern criteria. The dateValue is defined in the JDBC date format. If a dateField and dateValue are specified, all items that match the pattern criteria and are older than the date specified for the dateField will be deleted. Items that match the pattern criteria but are newer than the date specified will not be deleted.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
dTaskID	The task ID. This task ID is used for checking the status of the batch deletion by pattern.
ermStatusMessage	Error message.

Parameter	Description
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
DeleteByPattern(String dSource, String field, String pattern, String operator,
String dateField, String dateValue, IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
String dTaskID
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

DELETE_EXTERNAL

This service is used to delete items from URM individually. This service is used in the following process:

❖ Deleting Individual Items (page 3-19)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
idKey (localdata)	The adapter's key field value.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
DeleteExternal (String dSource, String idKey, IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
```

```
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

DELETE_MULTIPLE_EXTERNAL

This service is used to delete items from URM in a batch. This service is used in the following process:

❖ Deleting Items in Batch (page 3-17)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
externalRecordsList	A content server result set. The result set column contains the adapter's key column.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
dTaskID	The task ID. This task ID is used for checking the status of the batch checkin.
ermStatusMessage	Error message.

Parameter	Description
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
DeleteMultipleExternal (String dSource, IdKeyList externalRecordsList,
IdcProperty[] extraProps)
```

Returns

```
String dTaskID
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdKeyList` (page A-101)
`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

EDIT_EXTERNAL

This service is used to update metadata for a single item in URM. This service is used in the following process:

❖ Updating Individual Items (page 3-10)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
propertyList (localdata)	The adapter's metadata field value pairs. Please note the following important considerations: • **xCategoryID** and **xFolderID**—a record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error. • Neither dDocName nor dLongName can be updated.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.

Parameter	Description
StatusMessage	Content Server message.

WSDL Service

Function

`EditExternal (String dSource, IdcProperty[] editValues, IdcProperty[] extraProps)`

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

EDIT_MULTIPLE_EXTERNAL

This service is used to update the URM metadata for items in a batch. This service is used in the following process:

- ❖ Updating Items in Batch (page 3-7)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
externalRecordsMetadataList	A content server result set of propertyLists. Please note the following important considerations: • **xCategoryID** and **xFolderID**—a record can only be in a category or a folder, but not both. You can map both the xCategoryID and xFolderID fields, however for each record you can only send a value for one or the other. If you include both an xCategoryID and an xFolderID value for a record, URM will send an error. • Neither dDocName nor dLongName can be updated.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 102– Retrieval error.
dTaskID	The task ID. This task ID is used for checking the status of the batch update.

Parameter	Description
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
UpdateMultipleExternal (String dSource,
ExternalMetaDataList externalRecordsMetadataList, IdcProperty[] extraProps)
```

Returns

```
String dTaskID
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`ExternalMetaDataList` (page A-100)
`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

GET_EXTERNAL_DEFAULT_FIELDS

This service is used to get default URM external metadata fields. These fields must be mapped to the adapter's fields.

This service is used in the following process:

- Adapter Registration (page 2-1)

This service can be used in the following processes:

- Updating External Field Mappings (page 2-7)
- Adding External Custom Fields (page 2-9)
- Updating External Custom Fields (page 2-13)

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 301 – Setup error.
defaultExternalFields	A content server result set that lists all default URM fields that adapters can map.
securityGroups	The option list values for the dSecurityGroup field.
docAccounts	The option list values for the dDocAccount field.
docTypes	The option list values for the dDocTypes field.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
GetExternalDefaultFields (IdcProperty[] extraProps)
```

Returns

```
ExternalCustomMetaDefinition defaultExternalFields
int ermStatusCode
String ermStatusMessage
ExternalField securityGroups
ExternalField docAccounts
ExternalField docTypes
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`ExternalCustomMetaDefinition` (page A-100)
`ExternalField` (page A-100)
`StatusInfo` (page A-101)

GET_EXTERNAL_FREEZE_LIST

If there are new freezes, this service is used to request a list of frozen items from URM. This service is used in the following process:

❖ Performing Holds/Freezes (page 4-10)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.

Parameter	Description
beginDate	The time when the adapter last processed freezes. Please note the following important considerations: • If it is the adapter's first call to the GET_EXTERNAL_FREEZE_LIST service, the adapter should leave beginDate empty. • If the adapter has called the GET_EXTERNAL_FREEZE_LIST service before, it should have saved the endDate it received from URM the last time it called the GET_EXTERNAL_FREEZE_LIST service. The adapter must use this endDate as the beginDate for the next call to the GET_EXTERNAL_FREEZE_LIST service. • The beginDate and endDate are defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 102 – Retrieval error.
externalFreezeList	A content server result set that lists all frozen items.
hasMore	false – No more frozen items. true – More frozen items.

Parameter	Description
endDate	The time that the freeze query ran. Please note the following important considerations: • If there is no error with the service request, URM will return an externalFreezeList result set listing a block of frozen items, an endDate identifying when the query ran, and a Boolean flag (hasMore) indicating if there are more freeze items. When the adapter is finished processing the block of items, it uses the endDate that was sent with the block of items as the retrieveDate to mark the items as frozen.If there are more freeze items, the adapter calls the GET_EXTERNAL_FREEZE_LIST service again, using the same beginDate as it did initially. The adapter then receives the next block of items and a new corresponding endDate, which becomes the retrieveDate for this next block of items. When there are no more freeze items, the adapter must save the endDate for the last block of items received, as the adapter must use this endDate as the beginDate the next time it calls the CHECK_PENDING_EXTERNAL_TASK and GET_EXTERNAL_FREEZE_LIST services. • The beginDate and endDate are defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
GetExternalFreezeList (String dSource, String beginDate, IdcProperty[] extraProps)
```

Returns

```
FreezeList externalFreezeList
```

```
String endDate
boolean hasMore
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`FreezeList` (page A-100)
`StatusInfo` (page A-101)

GET_EXTERNAL_TABLE_FIELDS

This service is used to retrieve the adapter's mapped fields and custom fields. The columns will be the adapter's column names.

This service is used in the following processes:

- Declaring Items in Batch (page 3-2)
- Declaring Individual Items (page 3-5)
- Updating Items in Batch (page 3-7)
- Updating Individual Items (page 3-10)

This service can be used in the following processes:

- Updating External Field Mappings (page 2-7)
- Adding External Custom Fields (page 2-9)
- Updating External Custom Fields (page 2-13)
- Checking Items into URM (Internal Checkin) (page 3-22)

Required Parameter

This parameter must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.

Optional Parameter

This optional parameter may be specified:

Parameter	Description
customFieldsOnly	1 – Only custom fields to be in the results set. 0 – Both mapped fields and custom fields.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 301 – Setup error.
externalTableFields	A content server result set. Specify any adapter-specific metadata fields. Return parameters for option lists are optional.
securityGroups	The option list values for the dSecurityGroup field.
docAccounts	The option list values for the dDocAccount field.
docTypes	The option list values for the dDocTypes field.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

Example externalTableFields Result Set

```
@ResultSet externalTableFields
14
dName
dCaption
dType
dIsRequired
dIsEnabled
dIsSearchable
```

```
dIsOptionList
dDefaultValue
dOptionListKey
dOptionListType
dHides
dRequires
dOrder
dIsDisplayOnly
@end
```

WSDL Service

Function

```
GetExternalTableFields (String dSource, boolean customFieldsOnly,
IdcProperty[] extraProps)
```

Returns

```
ExternalCustomMetaDefinition externalTableFields
int ermStatusCode
String ermStatusMessage
ExternalField securityGroups
ExternalField docAccounts
ExternalField docTypes
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`ExternalCustomMetaDefinition` (page A-100)
`ExternalField` (page A-100)
`StatusInfo` (page A-101)

GET_EXTERNAL_UNFREEZE_LIST

If there are new unfreezes, this service is used to request a list of unfrozen items from URM. This service is used in the following process:

❖ Removing Holds/Freezes (page 4-14)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
beginDate	The time when the adapter last processed unfreezes. Please note the following important considerations: • If it is the adapter's first call to the GET_EXTERNAL_UNFREEZE_LIST service, the adapter should leave beginDate empty. • If the adapter has called the GET_EXTERNAL_UNFREEZE_LIST service before, it should have saved the endDate it received from URM the last time it called the GET_EXTERNAL_UNFREEZE_LIST service. The adapter must use this endDate as the beginDate for the next call to the GET_EXTERNAL_UNFREEZE_LIST service. • The beginDate and endDate are defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 102 – Retrieval error.
externalUnFreezeList	A content server result set that lists all unfrozen items.
hasMore	false – No more unfrozen items. true – More unfrozen items.

Parameter	Description
endDate	The time that the unfreeze query ran. Please note the following important considerations: • If there is no error with the service request, URM will return an externalUnFreezeList result set listing a block of unfrozen items, an endDate identifying when the query ran, and a Boolean flag (hasMore) indicating if there are more unfreeze items. When the adapter is finished processing the block of items, it uses the endDate that was sent with the block of items as the retrieveDate to mark the items as unfrozen.If there are more unfreeze items, the adapter calls the GET_EXTERNAL_FREEZE_LIST service again, using the same beginDate as it did initially. The adapter then receives the next block of items and a new corresponding endDate, which becomes the retrieveDate for this next block of items. When there are no more unfreeze items, the adapter must save the endDate for the last block of items received, as the adapter must use this endDate as the beginDate the next time it calls the CHECK_PENDING_EXTERNAL_TASK and GET_EXTERNAL_UNFREEZE_LIST services. • The beginDate and endDate are defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
GetExternalUnfreezeList (String dSource, String beginDate,
IdcProperty[] extraProps)
```

Returns

```
FreezList externalUnFreezeList
String endDate
boolean hasMore
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`FreezeList` (page A-100)
`StatusInfo` (page A-101)

GET_FILE_PLAN

This service is used to download, from URM, only aspects of the retention schedule that a particular user can view. This service is used in the following process:

❖ Downloading Parts of the Retention Schedule for Viewing (page 5-3)

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
filePlan	XML file name. File contains partial retention schedule.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
GetFilePlan (IdcProperty[] extraProps)
```

Returns

```
IdcFile filePlan
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcFile` (page A-101)
`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

GET_FILE_PLAN_ALL

This service is used to download, from URM, the entire retention schedule as a single XML block. This service is used in the following process:

❖ Downloading the Entire Retention Schedule (page 5-2)

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error.* 101 – Data error.
filePlan	XML file name. File contains entire retention schedule.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
GetFilePlanAll (IdcProperty[] extraProps)
```

Returns

```
IdcFile filePlan
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcFile` (page A-101)
`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

GET_LIFECYCLE_FOR_EXTERNAL_ITEM

This service is used to request the lifecycle for a specific item from URM. This service is used in the following process:

- Requesting the Lifecycle for an Item (page 5-5)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
idKey (localdata)	The adapter's key field value.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.

Parameter	Description
itemDispositions	List of disposition information for the item.
categoryInfo	Category information for any categories in itemDispositions.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
GetLifeCycleForExternalItem (String dSource, String idKey,
IdcProperty[] extraProps)
```

Returns

```
Dispositions itemDispositions
CategoryInfo categoryInfo
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`Dispositions` (page A-99)
`CategoryInfo` (page A-99)
`StatusInfo` (page A-101)

GET_MAXIMUM_FIELD_LENGTHS

This service is used to determine the maximum field length for each text type. This service is used in the following process:

❖ Adding External Custom Fields (page 2-9)

Required Parameters

No parameters are required.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.

maximumFieldLengths	A content server result set that lists the maximum field length for each text type.
	This result set contains two columns:
	`type`
	`length`
	The following text types are returned: Text, BigText, and Memo. The length value for each text type represents the maximum length for any field of that type.

ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
GetMaximumFieldLengths (IdcProperty[] extraProps)
```

Returns

```
FieldLengths maximumFieldLengths
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`FieldLengths` (page A-100)
`StatusInfo` (page A-101)

GET_SEARCH_REQUEST

This service is used to request a list of pending search criteria from URM. This service is used in the following process:

- ❖ Performing Federated Searches (page 4-1)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

Parameter	Description
searchRequests	The content server search query. Please note the following important considerations: • Items in the searchRequest may be a combination including fields, operators, values. • Items may also be conjunctions. • Valid conjunctions are AND and OR

Example searchRequest Query Format

The following is an an example searchRequest query:

```
<query>

<item>
<field>FieldNameOne</field>
<op>equals</op>
<value>value</value>
</item>

<item>
<conj>AND</conj>
</item>

<item>
<field>FieldNameTwo</field>
<op>equals</op>
<value>value</value>
</item>

</query>
```

Operator Values

The following operators are used in the searchRequest query:

Operator	Description
equals	Exact match of the search criteria.
notEquals	Match of everything except exact match of search criteria.
hasAsSubstring	Match contains search criteria.
notHasAsSubstring	Match does not contain search criteria.

Operator	Description
beginsWith	Match starts with search criteria.
endsWith	Match ends with search criteria.
dateGE	Match items where the date is greater than or equal to the specified date.
dateLess	Match items where the date is less than the specified date.
numberGreater	Match items where the number is greater than specified number.
numberGE	Match items where the number is greater or equal than specified number.
numberLE	Match items where the number is less than or equal to specified number.
numberEquals	Match items where the number is equals to specified number.
numberLess	Match items where the number is less than specified number.
ftx	Match contains the value of a full boolean text search. The full text search function matches a natural language query against a text collection. The boolean full-text search capability supports the following operators: • A space between search terms matches items only if all terms are found. • A comma (,) between search terms matches a result if either term is found. • A minus (-) immediately in front of a term indicates that this word must not be present in any matches. • Parentheses are used to group search terms into subexpressions. • A phrase enclosed in double quotes ("), matches only items that contain the exact phrase.

WSDL Service

Function

```
GetSearchRequest (String dSource, IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`SearchRequests` (page A-101)
`StatusInfo` (page A-101)

INFO_EXTERNAL_ITEM

This service is used to request URM metadata for a specific item. This service is used in the following process:

❖ Requesting URM Metadata for an Item (page 5-4)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
idKey (localdata)	The adapter’s key field value.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
externalItemInfo	A content server result set that lists the URM metadata for the item. The columns will be the adapter’s column names.

Parameter	Description
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
InfoExternalItem (String dSource, String idKey, IdcProperty[] extraProps)
```

Returns

```
ExternalMetaDataList externalItemInfo
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

```
IdcProperty (page A-101)
ExternalMetaDataList (page A-100)
StatusInfo (page A-101)
```

LIST_EXTERNAL_APPROVED_DISP_ACTIONS

If there are new pending approved dispositions, this service is used to request a list of pending dispositions from URM. This service is used in the following process:

❖ Performing Dispositions (page 4-5)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.

Parameter	Description
beginDate	The time when the adapter last processed dispositions. Please note the following important considerations: • If it is the adapter's first call to the LIST_EXTERNAL_APPROVED_DISP_ACTIONS service, the adapter should leave beginDate empty. • If the adapter has called the LIST_EXTERNAL_APPROVED_DISP_ACTIONS service before, it should have saved the last endDate it received from URM. This endDate is then used as the beginDate for the next call to this service. • The beginDate and endDate are defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
approvedDispActionsList	A content server result set that contains a list of disposition actions that are approved.

Parameter	Description
endDate	The time that the disposition query ran. The adapter must save this endDate, as it will be used as the beginDate for the next call to the CHECK_PENDING_EXTERNAL_TASK and LIST_EXTERNAL_APPROVED_DISP_ACTIONS services. **Note:** The beginDate and endDate are defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

Example approvedDispActionsList Result Set

```
@ResultSet approvedDispActionsList
28
dDispositionID 6 32
dPreviousID 6 32
dCategoryID 6 100
dFolderID 6 100
dDispOrder 3 19
dDispAction 6 32
dDispPeriod 3 19
dDipsPeriodUnits 6 32
dDispEventTrigger 6 100
dDispCutoffCount 3 19
dDispTriggerType 6 32
dIsSystemDerived 1 8
dDispLocation 6 100
dDispLocation2 6 100
dDispReviewer 6 100
dDerivedTriggerType 6 32
```

```
dDerivedEventTrigger 6 100
dDerivedMonthDelay 3 19
dDerivedDayDelay 3 19
dDocAuthor 6 30
dTaskID 6 100
dIsPermanent 1 8
dSingleItemApproval
dAllowScheduling
dActionService1
dActionService1String
dActionService2
dActionService2String
@end
```

From the approvedDispActionsList result set, the adapter must maintain the dDispositionID. The dDispositionID is required when calling LIST_EXTERNAL_ITEMS_FOR_DISP_ACTION (page A-69).

The adapter might also want to maintain the dDispAction. The dDispAction might be required if you need to map disposition actions between URM and the adapter.

Delete and Scrub the following actions:

- wwRmaDeleteAllRevisions
- wwDeleteRevision
- wwRmaDestroy

Create an archive zip file for the following actions:

- wwRmaAccession
- wwRmaArchive
- wwRmaMove
- wwRmaTransfer

Note: For more information, see Performing Dispositions (page 4-5).

WSDL Service

Function

```
ListExternalApprovedDispActions (String dSource, String beginDate,
IdcProperty[] extraProps)
```

Returns

```
ApprovedDispActions approvedDispActionsList
```

```
String endDate
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`ApprovedDispActions` (page A-99)
`StatusInfo` (page A-101)

LIST_EXTERNAL_ITEMS_FOR_DISP_ACTION

This service is used to get a list of items that are subject to a specific disposition action. This service is used in the following processes:

❖ Performing Dispositions (page 4-5)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
dDispositionID	The disposition ID that was included in the approvedDispActionsList result set returned when LIST_EXTERNAL_APPROVED_DISP_ACTIONS (page A-65) was called.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 102 – Retrieval error.

Parameter	Description
approvedDispItems	A content server result set that lists items that are due for a specific disposition action. Please note the following important consideration: • URM will return dDispAction=wwRmaArchive for the following action types: Accession, Archive, Move, and Transfer. This identifies the action as an archive action, and the adapter must bundle the corresponding items into an archive zip file and call UPLOAD_EXTERNAL_ARCHIVE service.
hasMore	false – No more dispositions. true – More dispositions.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
ListExternalItemsForDispAction (String dSource, String dDispositionID,
IdcProperty[] extraProps)
```

Returns

```
DispositionItemList approvedDispItems
boolean hasMore
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

IdcProperty (page A-101)
DispositionItemList (page A-99)
StatusInfo (page A-101)

MARK_SELECTED_ITEMS_DISP_ACTION

This service is used to mark the disposition of items as complete. This service is used in the following process:

❖ Performing Dispositions (page 4-5)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
dDispositionID	The disposition ID that was included in the approvedDispActionsList result set returned when LIST_EXTERNAL_APPROVED_DISP_ACTIONS (page A-65) was called.
dispositionCompletedList	A content server result set that lists items the adapter disposed of successfully.
dispositionErrors	A content server result set that lists items that the adapter couldn’t dispose of due to errors.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
MarkSelectedItemsDispAction (String dSource, String dDispositionID,
DispositionCompletedList dispositionCompletedList,
DispositionErrors dispositionErrors, IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`DispositionCompletedList` (page A-99)
`DispositionErrors` (page A-99)
`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

MARK_SELECTED_ITEMS_FROZEN

This service is used to mark the freeze of items as complete. This service is used in the following process:

❖ Performing Holds/Freezes (page 4-10)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
retrieveDate	The endDate that was returned by URM for the corresponding block of items when the GET_EXTERNAL_FREEZE_LIST service was called. It is critical that the retrieveDate and endDate match, exactly, the endDate that was returned by URM for the corresponding block of items. **Note:** The retrieveDate is defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).

Parameter	Description
freezeCompletedList	A content server result set that lists items the adapter froze successfully.
freezeErrors	A content server result set that lists items that the adapter couldn't freeze due to errors. Please note the following important consideration: • **errorMessage**—This column gives a text description of the problem.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
MarkSelectedItemsFrozen (String dSource, FreezeCompletedList freezeCompletedList,
FreezeErrors freezeErrors, String retrieveDate, IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

FreezeCompletedList (page A-100)
FreezeErrors (page A-100)

IdcProperty (page A-101)
StatusInfo (page A-101)

MARK_SELECTED_ITEMS_UNFROZEN

This service is used to mark the unfreeze of items as complete. This service is used in the following process:

❖ Removing Holds/Freezes (page 4-14)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
retrieveDate	The endDate that was returned by URM for the corresponding block of items when the GET_EXTERNAL_UNFREEZE_LIST service was called. It is critical that the retrieveDate and endDate match, exactly, the endDate that was returned by URM for the corresponding block of items. **Note:** The retrieveDate is defined as the String object type, in the format of a timestamp representing the milliseconds since 1/1/1970 GMT. This should be a 13-digit number that starts with 11 (for example, 1163094936297).
unfreezeCompletedList	A content server result set that lists items the adapter unfroze successfully.
unfreezeErrors	A content server result set that lists items the adapter couldn't unfreeze due to errors. Please note the following important consideration: • **errorMessage**—This column gives a text description of the problem.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
MarkSelectedItemsUnfrozen (String dSource,
FreezeCompletedList unfreezeCompletedList, FreezeErrors unfreezeErrors,
String retrieveDate, IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`FreezeCompletedList` (page A-100)
`FreezeErrors` (page A-100)
`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

PING_SERVER

This service is used to determine if the URM server is running. This service is used in the following process:

❖ Pinging the URM Server (page 6-12)

Required Parameters

No parameters are required.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed. **Note:** A Failed status code indicates that there was a problem executing the service. However, this does not mean that the adapter failed to communicate with the URM server. Either a returned status code of 0 or 1 indicates that the adapter was able to communicate with the URM server.
StatusMessage	Content Server message.

WSDL Service

Function

`PingServer (IdcProperty[] extraProps)`

Returns

`StatusInfo statusInfo`

Complex Types

`StatusInfo` (page A-101)

RETURN_SEARCH_RESULTS

This service is used to upload search results to URM. This service is used in the following process:

❖ Performing Federated Searches (page 4-1)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
dSearchTaskID	The identifier (ID) for the requested search query.
searchItemCount	The number of items in the returned result set.
fileItemCount	Number of items that have a data file inside the .zip file.
firstBlock	true – Indicates this is the first block of search results for this criteria. false – Indicates this is not the first block of search results for this criteria. This should be reset for each unique dSearchTaskID.
hasMore	true – Indicates there are more items to be returned for this search.. false – Indicates there are no more items to be returned for this search and the results are finished.
searchResultsFile	The .zip file containing the search results. The .zip file contains the following items: • The metadata.xml file. This file contains the metadata for the search results to be returned. • Any files to be returned to the Content Server are sent inside the .zip file.

Example metadata.xml File Format

The metadata.xml file contains the metadata results for this block of search criteria. Each item returned in the search results is represented by <item> followed by the mapped fields for the item. Note the following considerations:

- Each item must include two required mapped fields. Required fields include the external mapped dDocTitle, and either the dDocName or dLongName (choose the field that is being used for the Unique Id). For more details on external mapped fields, see Updating External Field Mappings (page 2-7)
- It is recommended to include other external mapped fields for each item. Including this metadata will allow URM to display the information in the Search Result Detail Page.
- Unmapped fields may also be included with the item, as optional metadata. This information will stay with the search results, although may not be displayed in the URM Search Results Page
- If a file is being returned with the search results, the metadata.xml must include <filename> with the item. The filename should give both the name and exact path to the item within the zip file.
- If no file is being returned, <filename> must not be included.

In the following example of the metadata.xml format, there are three search results being returned. The first and third results are being returned with a file. The following is an an example searchRequest query:

```
<metadata>

<item>
<adapter_field1>ExternalMappedContentId</adapter_field1>
<adapter_field2>ExternalMappedContentTitle</adapter_field2>
<adapter_field3>value</adapter_field3>
<adapter_field4>value</adapter_field4>
<adapter_field5>value</adapter_field5>
<filename>filepath</filename>
</item>

<item>
<adapter_field1>ExternalMappedContentId</adapter_field1>
<adapter_field2>ExternalMappedContentTitle</adapter_field2>
<adapter_field3>value</adapter_field3>
<adapter_field4>value</adapter_field4>
<adapter_field5>value</adapter_field5>
</item>

<item>
<adapter_field1>ExternalMappedContentId</adapter_field1>
```

```
<adapter_field2>ExternalMappedContentTitle</adapter_field2>
<adapter_field3>value</adapter_field3>
<adapter_field4>value</adapter_field4>
<adapter_field5>value</adapter_field5>
<filename>filepath</filename>
</item>
</metadata>
```

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
ReturnSearchResults (String dSource, String dSearchTaskID, int searchItemCount,
int fileItemCount, boolean firstBlock, boolean hasMore, IdcFile searchResultsFile,
IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
```

Complex Types

StatusInfo (page A-101)

SET_DEFAULT_EXTERNAL_SECURITY_GROUP

This service is used during adapter registration to set the default security group. If a default security group is not set, the Public security group is used if a security group is not specified when declaring items. This service is used in the following process:

❖ Adapter Registration (page 2-1)

Required Parameters

These parameters must be specified:

Parameter	**Description**
dSource	The adapter identifier (ID) created when the adapter was registered.
dSecurityGroupName	The name of the default security group. This security group must exist in URM. It must be created in URM by the administrator manually.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	**Description**
ermStatusCode	0 – No error. 301– Setup error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
SetDefaultExternalSecurityGroup (String dSource, String dSecurityGroupName,
IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

SETUP_EXTRA_METADATA_FIELD

This service is used to activate the extra metadata field after registering the adapter. Calling this service creates an extra table with a dExtraMetaData column. It is then possible to pass a dExtraMetaData value during future item checkins or edits

It is currently recommended that you pass an XML string with extra metadata field mappings for dExtraMetaData. The dExtraMetaData string would be included along with all other field mapping data; this is just a special field that is not created or mapped.

This service is used in the following process:

❖ Setting Up the Extra Metadata Field (page 2-16)

Important: Once SETUP_EXTRA_METADATA_FIELD is called and the extra metadata field is created, it cannot be removed.

Required Parameter

This parameter must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 301 – Setup error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
SetupExtraMetadataField (String dSource, IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

TRANSFER_ITEM_TO_INTERNAL

This service is used to move an item that has been declared to URM and is being stored externally into the URM repository (perform an internal checkin on an item that has been declared to URM and is being managed externally), so that the item can be preserved and purged properly.

This service is used in the following processes:

- ❖ Transferring Items to URM (page 3-25)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
idKey (localdata)	The adapter's key field value.
primaryFile	The file to be uploaded to URM.

Important: If you are using profiles, you need to have mapped xRMProfileTrigger.

Optional Parameter

This optional parameter may be specified:

Parameter	Description
dDocAuthor	dDocAuthor is optional. If dDocAuthor is sent, it cannot be the mapped value; it must be dDocAuthor. If dDocAuthor is not sent, URM will use the author value in the external table for the item to be transferred.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
docUrl	The URL to the item in URM.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
TransferItemToInternal (String dSource, String idKey, String dDocAuthor,
IdcFile primaryFile, IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
String docUrl
StatusInfo statusInfo
```

Complex Types

IdcFile (page A-101)
IdcProperty (page A-101)
StatusInfo (page A-101)

UPDATE_EXTERNAL_CUSTOM_FIELDS

This service is used to update external custom fields after registering the adapter.

This service is used in the following process:

❖ Updating External Custom Fields (page 2-13)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.
externalCustomMetaDefinition	A content server result set. Specify any adapter-specific metadata fields. Please note the following important considerations: • If a field does not exist already, you will receive an error. • **dOptionListType**—only the following values are valid: combo, multi, or strict • **dType—**only the following values are valid: Date, Int, Text, BigText, or Memo • **dIsDisplayOnly—**indicates if the field is a placeholder field. If the field is defined as a placeholder field, URM does not create the field in the source table.

Example externalCustomMetaDefinition Result Set

The following is an example externalCustomMetaDefinition result set in an .hda file format. Please note that this format is not required; other formats can be used.

```
<?hda version="7.5.1 (050330)" jcharset=UTF8 encoding=utf-8?>
@Properties LocalData
blFieldTypes=
blDateFormat=M/d/yy {h:mm[:ss] {aa}[zzz]}!mAM,PM!tAmerica/Chicago
@end
@ResultSet externalCustomMetadataDefinition
14
dName
dCaption
dType
dIsRequired
dIsEnabled
dIsSearchable
dIsOptionList
dDefaultValue
dOptionListKey
dOptionListType
dHides
dRequires
dOrder
dIsDisplayOnly
ExternalComments
External Comments
Memo
false
true
true
false

5
false
Department
Department
Text
true
true
true
false

6
false
Color
Color Option List Field
Text
false
```

```
true
true
true
Red
ColorList
strict

7
false
ExternalBigText
External Big Text Field
BigText
false
true
true
false

8
false
@end
```

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 301 – Setup error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
UpdateExternalCustomFields (String dSource,
ExternalCustomMetaDefinition externalCustomMetaDefinition,
IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

ExternalCustomMetaDefinition (page A-100)
IdcProperty (page A-101)
StatusInfo (page A-101)

UPDATE_EXTERNAL_FIELD_MAPPING

This service is used to update external field mappings after registering the adapter.

This service is used in the following process:

❖ Updating External Field Mappings (page 2-7)

Required Parameters

These parameters must be specified:

Parameter	Description
dSource	The adapter identifier (ID) created when the adapter was registered.

Parameter	Description
externalFieldMap	A content server result set with the columns rmField, externalField, and externalCaption. Specify the fields mapping between URM fields and the adapter’s fields. Please note the following important considerations: • If rmField already exists, the mapped externalField and externalCaption will be updated. Making changes to external field mappings never results in database changes; it only affects the metadata mapping for future items. The metadata mapping for any existing items will not be changed by updating the external field mappings. • If you want to maintain the original field data when it needs to be truncated, consider creating a custom field, with the same name as the mapped field, that is long enough to hold the original data. For example, consider you have a Subject field that you map to dDocTitle, which is indicated as a truncated field. You could create a custom field named Subject. Then you only need to send the Subject field value once, as URM will truncate the data if it is greater than the length of the field for dDocTitle. The full value will still be in the Subject field.

Example externalFieldMap Result Set

The following is an example externalFieldMap result set in an .hda file format. Please note that this format is not required; other formats can be used.

```
<?hda version="7.5.1 (050330)" jcharset=UTF8 encoding=utf-8?>
@Properties LocalData
blFieldTypes=
blDateFormat=M/d/yy {h:mm[:ss] {aa}[zzz]}!mAM,PM!tAmerica/Chicago
@end
@ResultSet externalFieldMap
3
rmField
externalField
externalCaption
dDocName
exDocName
EX Content ID
dSecurityGroup
exSecurityGroup
EX Security Group
dDocAccount
exDocAccount
EX Account
dDocTitle
exDocTitle
EX Title
dCreateDate
exCreateDate
EX Create Date
dDocAuthor
exAuthor
EX Author
xExternalLocation
exDocLocation
EX Doc Location
dLongName
exLongName
EX Long Name
dDocType
exDocType
EX Type
xIsRecord
exIsRecord
EX Is Record
xCategoryID
exCategoryID
EX Category
xFolderID
exFolderID
```

```
EX Folder
@end
```

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 301 – Setup error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

WSDL Service

Function

```
UpdateExternalFieldMapping (String dSource, ExternalFieldMap externalFieldMap,
IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

ExternalFieldMap (page A-100)
IdcProperty (page A-101)
StatusInfo (page A-101)

UPLOAD_EXTERNAL_ARCHIVE

When processing dispositions, if the action is a move action:

- Accession (wwRmaAccession)
- Archive (wwRmaArchive)
- Move (wwRmaMove)

- Transfer (wwRmaTransfer)

the adapter bundles the corresponding items into an archive zip file and calls this service to upload the archive zip file to URM. The zip file of items must contain an *index.hda* file. For an example, see Example index.hda File (page A-94).

This service is used in the following processes:

❖ Performing Dispositions (page 4-5)

Required Parameters

These parameters must be specified:

Parameter	**Description**
dSource	The adapter identifier (ID) created when the adapter was registered.
dDispositionID	The disposition ID that was included in the approvedDispActionsList result set returned when LIST_EXTERNAL_APPROVED_DISP_ACTIONS (page A-65) was called.
dispositionCompletedList	A content server result set that lists items the adapter disposed of successfully.
dispositionErrors	A content server result set that lists items that the adapter couldn't dispose of due to errors. Please note the following important considerations: • **errorMessage**—this column gives a text description of the problem. • The errorCode column was removed in version 7.1.4, as there was no need for it.
archiveFile	A zip file containing the item files to be uploaded to URM for archival and an *index.hda* file. For an example *index.hda* file, see Example index.hda File (page A-94).

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
dTaskID	The task ID. This task ID is used for checking the status of the archive upload.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

Example index.hda File

An *index.hda* file must be included in the zip file of document items sent with the UPLOAD_EXTERNAL_ARCHIVE service. The following is an example of what the *index.hda* file might look like:

```
@ResultSet FileMap
2
filename
idKey
one.doc
one
ac
two.doc
two
jj
@end
```

WSDL Service

Function

```
UploadExternalArchive (String dSource, String dDispositionID,
DispositionCompletedList dispositionCompletedList,
DispositionErrors dispositionErrors, IdcFile archiveFile,
IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
String ermStatusMessage
String dTaskID
StatusInfo statusInfo
```

Complex Types

`DispositionCompletedList` (page A-99)
`DispositionErrors` (page A-99)
`IdcFile` (page A-101)
`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

UPLOAD_EXTERNAL_LOG_FILE

This service is used to upload external log files to URM. This service is used in the following processes:

❖ Uploading External Log Files (page 6-11)

Required Parameters

These parameters must be specified:

Parameter	Description
logID	The ID for the log file.
logFile	A zip file containing the external log files to be uploaded to URM and an *index.hda* file. For an example *index.hda* file, see Example index.hda File (page A-97). Please note the following important considerations: • By default, URM will store a maximum of 30 external log files. URM will upload up to 30 external log files, if they are newer than any files already stored and if they do not already exist. You can change the maximum number of external log files that URM will store by adding the `MaxNumberOfExternalLogFiles` configuration variable to the *config.cfg* file for the URM Server. • You can view external log files in the URM interface by navigating to Administration—Log Files—URM Logs.

Returned Parameters

The adapter expects these parameters to be returned:

Parameter	Description
ermStatusCode	0 – No error. 101 – Data error.
ermStatusMessage	Error message.
StatusCode	Content Server status code: 0 – Processed successfully. -1 – Failed.
StatusMessage	Content Server message.

Example index.hda File

An *index.hda* file must be included in the zip file of external log files sent with the UPLOAD_EXTERNAL_LOG_FILE service.

If you have a zip file called *logFiles.zip*, which contains two external log files, *IdcLog01.htm* and *IdcLog02.htm* (which are formatted properly for display in the URM interface), you would include in this zip file an *index.hda* file that would look as follows:

```
@ResultSet LogFiles
2
filename
date
IdcLog01.htm
{ts '2010-08-31 01:36:49.176'}
IdcLog02.htm
{ts '2010-08-30 01:36:49.176'}
@end
```

You must include the timestamp for each external log file so that URM can determine if the log file already exists and if it is newer than those it has stored already.

WSDL Service

Function

```
UploadExternalLogFile (String logID, IdcFile logFile, IdcProperty[] extraProps)
```

Returns

```
int ermStatusCode
```

```
String ermStatusMessage
StatusInfo statusInfo
```

Complex Types

`IdcFile` (page A-101)
`IdcProperty` (page A-101)
`StatusInfo` (page A-101)

WSDL Return Parameter Types (Complex Types)

Complex Type Name	Elements

ApprovedDispActions	`String dDispositionID`
	`String dDispAction`
	`String dFolderID`
	`String dCategoryID`

CategoryInfo	`String dCategoryID`
	`String dCategoryName`
	`String dCategoryDescription`
	`String dSecurityGroup`
	`String dDispositionType`

DispositionCompletedList	`String idKey`
	`String dMarkCompleteDate`

DispositionErrors	`String idKey`
	`String errorMessage`

DispositionItemList	`String idKey`
	`String dApprovedDate`
	`String dLocalParameters`

Dispositions	`Date dActionDate`
	`String dDispEventTrigger`
	`String dDispAction`
	`String dDispReviewer`

ErrorList	`String idKey`
	`int errorCode`
	`String errorMessage`

Complex Type Name	Elements

ExternalCustomMetaDefinition	`String dName`
	`String dCaption`
	`String dType`
	`boolean dIsRequired`
	`boolean dIsEnabled`
	`boolean dIsSearchable`
	`boolean dIsOptionList`
	`String dDefaultValue`
	`String dOptionListKey`
	`String dOptionListType`
	`String dHides`
	`String dRequires`
	`int dOrder`
	`boolean dIsDisplayOnly`

ExternalField	`String dValue`
	`String dKey`
	`String dOption`
	`String dTooltip`

ExternalFieldMap	`String rmField`
	`String externalField`
	`String externalCaption`

ExternalMetaDataList	`IdcProperty[] metadataValues`

FieldLengths	`String type`
	`int length`

FreezeCompletedList	`String idKey`

FreezeErrors	`String idKey`
	`String errorMessage`

FreezeList	`String idKey`

Complex Type Name	Elements

IdcFile	`String fileName`
	`byte[] fileContent`

IdcProperty	`String name`
	`String value`

IdKeyList	`String idKey`

SearchRequests	`String dSearchTaskID`
	`int maxNumItems`
	`int maxNumFiles`
	`String searchQuery`

StatusInfo	`int statusCode`
	`String statusMessage`

TaskList	`String dTaskID`

TaskListStatus	`String dTaskID`
	`boolean isValidTaskID`
	`boolean isTaskCompleted`
	`boolean hasError`

Appendix

B

THIRD PARTY LICENSES

OVERVIEW

This appendix includes a description of the Third Party Licenses for all the third party products included with this product.

APACHE SOFTWARE LICENSE

W3C® SOFTWARE NOTICE AND LICENSE

ZLIB LICENSE

GENERAL BSD LICENSE

GENERAL MIT LICENSE

UNICODE LICENSE

MISCELLANEOUS ATTRIBUTIONS

Index