



Provider Hands On Lab









Agenda

Tuesday, November 4th, 2003 – Provider Hands On Lab

11/4/2003	9:00 AM	1:00	10:00 AM	Provider Accounts
11/4/2003	10:00 AM	0:30	10:30 AM	Ingest
11/4/2003	10:30 AM	0:15	10:45 AM	Break
11/4/2003	10:45 AM	0:30	11:15 AM	More Ingest
11/4/2003	11:15 AM	0:30	11:45 AM	Metadata Update
11/4/2003	11:45 AM	1:30	1:15 AM	Lunch
11/4/2003	1:15 PM	1:30	2:45 PM	Provider Order Mgmt Svc
11/4/2003	2:45 PM	0:15	3:00 PM	Break
11/4/2003	3:00 PM	1:30	4:30 PM	Access Control



Global Science & Technology, Inc.

2 



Provider Account Service Hands-on Session









Becoming a Provider: The First Step

- **Submit a provider application**
 - A temporary provider id will be assigned to this provider in return message.
 - This temporary number should be referenced in any future communication with the ECHO staff
- **Communicate with the ECHO staff to set up tables, ftp account space, etc**

• File: 01_SubmitProviderApplication.xml



Global Science & Technology, Inc.

4 



What Happens Next?

- **A dialog between the provider and the ECHO staff**
 - Establishes key information ECHO needs
 - Do you support ordering, cancelling, quoting?
 - What coordinate system do you use?
 - What userId would you like to represent this provider?
- **This usually takes awhile to complete**



Global Science & Technology, Inc.

5 



Let's Log-In

- Since we don't have the time for all of that, let's use one of the providers we have already set up.

- **Login as: DemoUser#**
- **Set provider context as: Demo_Provider#**



Global Science & Technology, Inc.

6 

Provider Contacts

- A provider can create multiple contacts, each with separate address and telephone information
 - Billing contact may be in a different geographic location than the Shipping contact
- Let's add some contacts to the provider
 - A shipping contact
 - A billing contact
- File: 02_AddContact.xml



Presenting Contacts

- You can present all the contacts for a provider or you can present a specific contact
- Note: ECHO converts all contact names to lower case. 'Billing Contact' becomes 'billing contact'
- File: 03_PresentContact.xml



Updating Contacts

- Addresses and telephone numbers change over time.
- Staff change over time
- Let's change the name of the Billing Contact
- File: 03_UpdateContact.xml



Presenting Contacts

- Let's present the contacts again
- You should see AJ PennyPacker instead of Bob Sacamono as the billing contact
- File: 04_PresentContact.xml



Presenting a Specific Contact

- We previously displayed the contact information for all the contacts
- This time we are interested in presenting a specific contact
- File: 05_PresentContact.xml



Deleting Contacts

- Deleting contacts is easy - simply specify the contact role (shipping contact for example)
- File: 06_DeleteContact.xml



Verify the Contacts

- Let's verify that the contact was actually removed by presenting all the contact information
- The 'shipping contact' should no longer exist
- File: 07_PresentContact.xml



Presenting all Information for a Provider

- Contacts are the primary data member for a provider
 - They are managed via the 'Contact transactions'
- The organization name of a provider can be presented via the PresentProviderInfo transaction
 - Take note that the current organization name is 'Demo Provider 1'
- File: 08_PresentProviderInfo.xml



Updating the Organization

- Although unlikely, a provider may wish to change it's organization name.
- File: 09_UpdateProviderInformation.xml



Verify the Organization

- Let's verify that the organization was actually changed.
- It should be 'Vandalay Industries'
- File: 10_PresentProviderInfo.xml



Clean-Up

- Up to this point we have changed:
 - The contacts for the provider
 - The organization name
 - The password the provider uses to login
- Let's clean-up those changes before moving on
- File: 12_UpdateContact.xml
- File: 13_AddContact.xml
- File: 14_UpdateProviderInfo.xml



Provider Policies

- Primarily used to tell ECHO how often and how frequently it should attempt to connect to a provider
- Like contacts, you can present a specific policy, or all policies
- Let's present all policies
- File: 16_PresentPolicyDefinition.xml



Presenting One Policy Definition

- Let's focus on one specific policy definition: max attempts to quote order
- What does this mean?
 - This policy is a member of the 'Retry Behavior' category
 - It must be present (min occurs and max occurs are both 1)
 - It is not encrypted
 - The value must be between 1 and 1000 inclusive
- File: 17_PresentPolicyDefinition.xml



So What Is Selected?

- Let's use the PresentPolicySelections transaction to see how many times ECHO should retry quote requests
- You will find no policy selections, meaning the provider has not told ECHO how many times to retry quote requests
- File: 18_PresentPolicySelections.xml



Setting a Policy

- Let's configure ECHO to retry quotes 4 times before giving up
- File: 19_SetPolicySelection.xml



Verify the Selection

- Re-run the PresentPolicySelections transaction.
- You have now told ECHO how many times to try to send quote requests to you before giving up
- File: 20_PresentPolicySelections.xml



Grant Provider Access

- Grant the specified users access to a provider
- File: 21_GrantProviderAccess.xml



For the Future

- The same mechanism is used for specifying the delay between retry attempts





Ingest Hands On Session









Collection DTD

- **Date Format**
- **Spatial**
- **Required Elements**
 - See sample XML file
- **You should always ingest the collection metadata before you ingest any of the granules in the collection**






Granule DTD

- **Date Format**
- **Spatial**
- **Required Elements**
 - See sample XML file
- **Reminder: You must submit the collection metadata before you put in any granules**

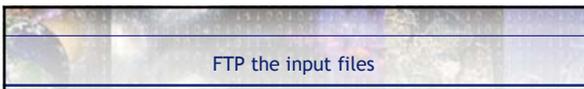





Browse DTD

- **Browse internal file name**
- **Browse file size**
- **Browse deletion not implemented**

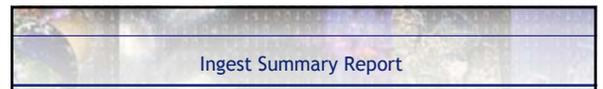


FTP the input files

- **FTP login user ID and Password**
- **Data directory**
 - Data
 - collection
 - granule
 - browse
 - update
 - valids



Ingest Summary Report

- **List of files processed (split files)**
- **Processing time stamp**
- **Total collections/granules Processed**
 - Accepted collection/granule
 - Inserted, replaced, deleted
 - Rejected collection/granule
 - out of date
 - duplicated
 - With bad data
 - Collections/Granules with invalid spatial data
 - Non-exists collection/granule for deletion
 - Granule associates with the collection that is not registered with ECHO
- **Sample File**




Error Input File Summary Report

- List of files rejected due to the following problems
 - The file does not have XML declaration line "<?xml...>" as the first line
 - The XML file be placed in wrong input directory
 - Fail to be validated against it's DTD
- Sample file



Insertion/Replacement Collection/Granule

- Using TAG indication
- Using insert/update date/time indication
- Replacement requirement (keep unique identifier)
- ECHO Ingest Handling Strategy
 - Out of date replacement: ignored
 - Duplicated records: take first entry that is most up to date
 - Automatic handling of insertion/replacement: If record exists then do replacement else do insertion



Getting Started

- FTP ingest file to orval.gsfc.nasa.gov
 - Open a dos window
 - Type this into the URL:
 - cd c:\the file path
 - ftp orval.gsfc.nasa.gov
 - Type demo_provider# as User, anything as password
 - Navigate to this directory:
 - cd data/collection or cd data/granule
 - Put Ingest.xml
 - The Ingest script will automatically pick it up within 5 minutes
- Open up another browser window and point to <http://mail.yahoo.com> and log in as demo_provider#
 - This is where reports from ingest will be delivered



Example 1 Submit input files for collection/granule insertion

- Goal: To observe what happens when provider submits the collection/granule for insertion
- Step 1: Check initial condition
 - Take a look at ECHO for all your collections
 - Submit 01_GetAllCollection.xml, 02_GetMOD01Collection.xml, 03_GetMOD01Granules.xml through the test harness
 - Note the number of hits for the 1st and 3rd queries
- Step 2: Ingest some data
 - Submit input xml files
 - FTP over 04_OneCollection.XML, 05_ManyCollections.XML and 06_Granules.XML
 - Make sure you put the collection files into the Collection folder and the granule file into the Granule folder
 - A cron job normally checks every five minutes for data
 - ECHO will move the data into the _in folder when it finds it



Example 1 Submit input files for collection/granule insertion

- Step 3: Get notified of ingest completion
 - Wait for ingest summary report from ECHO
 - Insertion information will be shown
 - Email will be sent to demo_provider# email account
 - Log into <http://mail.yahoo.com> with demo_provider# account
- Step 4: Validate the change occurred
 - Take a look at ECHO for all your collections
 - Using 07_GetAllCollection.xml, 08_GetMOD01Collection.xml, 09_GetMOD01Granules.xml
 - The total number of collections and granules should have increased



Example 2 Submit input files for collection update

- Goal: To observe what happens when provider submits a collection for update
- Step 1: Examine initial state
 - Take a look at ECHO for a specific collection
 - Use 01_GetMOD01Collection.xml to submit the query
- Step 2: Ingest the update file
 - Submit input xml files
 - FTP over 02_OneCollectionChanged_exp2.XML
 - CollectionState will be modified from "In Work" to "Completed"
 - Note that when updates are performed via the ingest mechanism, the entire metadata record must be provided
 - ECHO overwrites the existing record with the new record in its entirety



Example 2 Submit input files for collection update

- **Step 3: Get notified of ingest completion**
 - Wait for ingest summary report from ECHO
 - Replacement information shown
- **Step 4: Validate update occurred**
 - Take a look at ECHO again for this specific collection
 - Use 03_GetMOD01Collection.xml to submit the query
 - CollectionState should be "Completed"



Example 3 Submit input files for collection update with out of date record

- **Goal: To observe what happens when provider submits the collection for update with out of date information**
 - ECHO checks the Provider's timestamps and will not ingest data that is older than previously ingested data
- **Step 1: Examine initial condition**
 - Take a look at ECHO for a specific collection
 - Use 01_GetMOD01Collection.xml to submit the query
 - Check setting of Collection State
- **Step 2: Ingest the update file**
 - Submit input xml files
 - FTP over 02_OneCollectionChanged_exp3.XML
 - This file contains a last update time earlier than last submission and the Collection state is set to "Initialize"



Example 3 Submit input files for collection update with out of date record

- **Step 3: Get notified of ingest completion**
 - Wait for ingest summary report from ECHO
 - Out of date replacement information shown
- **Step 4: Evaluate**
 - Take a look at ECHO again for this specific collection
 - Use 03_GetMOD01Collection.xml to submit the query
 - No change has been made in ECHO
 - CollectionState has not been changed to "Initialize"



Example 4 Submit input files for collection update with duplicated records

- **Goal: To observe what happens when provider submits the collection for update with duplicated records**
- **Step 1: Check initial condition**
 - Take a look at ECHO for a specific collection
 - Using 01_GetMOD01Collection.xml
- **Step 2: Ingest the updated information**
 - Submit input xml files
 - FTP over 02_OneCollectionChanged_exp4.XML
 - This file contains duplicated collection entries
 - The earlier one is marked with a CollectionState of "Initialize"
 - The later one is marked with a CollectionState of "Completed"



Example 4 Submit input files for collection update with duplicated records

- **Step 3: Get notified of ingest completion**
 - Wait for ingest summary report from ECHO
 - Duplication information shown
- **Step 4: Validate most recent data is updated**
 - Take a look at ECHO again for this specific collection
 - Use 03_GetMOD01Collection.xml to submit the query
 - The one that has the most up to date information is ingested into ECHO (CollectionState = "Completed")



Example 5 Submit collection updates with violation of unique identifier rule

- **Goal: To observe what happens when provider submits the collection for update with collection's unique identifier rule been violated**
 - The scenario is that an update to an existing collection is what is intended, but due to a mistake, a new one is created instead
- **Step 1: Examine initial conditions**
 - Take a look at ECHO for a specific collection
 - Use 01_GetMOD01Collection.xml to submit a query
 - Version number is set to 3
- **Step 2: Ingest updated metadata files**
 - Submit input xml files
 - FTP over 02_OneCollectionChanged_exp5.XML
 - With version number set to 4



Example 5 Submit collection updates with violation of unique identifier rule

- **Step 3: Get notified of ingest completion**
 - Wait for ingest summary report from ECHO
 - Insertion information shown
- **Step 4:**
 - Take a look at ECHO again for this specific collection
 - Use 03_GetMOD01Collection.xml to submit a query
 - There has been no change to the existing collection
 - Use 04_GetAllCollections.xml to submit a query
 - A new collection been added to ECHO (Version 004)



Deletion of Collections and Granules

- Using TAG indication
- Using delete date indication
- Deleting a collection also deletes all the granules that belong to the collection



Example 6 Submit input files for granule deletion

- **Goal: To observe what happens when a provider submits a file for granule deletion**
- **Step 1: Examine Initial Conditions**
 - Take a look at ECHO for granules in a specific collection
 - Use 01_GetMOD01Granules.xml to submit a query
 - see SC:MOD01.003:16578351 and SC:MOD01.003:16578352
 - Extract the Catalog Item ID for SC:MOD01.003:16578351
 - Save this for the next step
 - Take note of the total number of hits
- **Step 2: Verify that an order can be created using this granule**
 - Submit an order with the Catalog Item ID you saved from Step 1 (SC:MOD01.003:16578351)
 - Use 02_CrOrderSC_MOD01.003_16578351_exp6.xml and substitute the correct CatalogItemID
 - Record and save the OrderID
 - Clean up order with 03_CleanupOrder.xml by pasting in the OrderID you just saved
 - We are just demonstrating that you can create the order here, so no need to save it



Example 6 Submit input files for granule deletion

- **Step 4: Ingest granule deletion file**
 - Submit input xml files
 - FTP over 04_GranuleDel_exp6.XML
 - This deletes the two granules listed above
- **Step 5: Get notified of ingest completion**
 - Wait for ingest summary report from ECHO
 - Deletion information should be shown
- **Step 6: Check that granules are gone**
 - Take a look at ECHO again for granule in this specific collection
 - Using 05_GetMOD01Granules.xml
 - Number of granules reduced by 2
 - Above two granules disappear (SC:MOD01.003:16578351 and SC:MOD01.003:16578352)



Example 6 Submit input files for granule deletion

- **Step 7: Check that you can't order the granule**
 - Submit an order with the CatalogItemID that you saved from Step 1
 - Use 06_CrOrderSC_MOD01.003_16578351_exp6.xml to create an order by pasting in the CatalogItemID
 - Catalog item deleted message is returned
 - Record and save the OrderID
 - Clean up order with 07_CleanupOrder.xml by pasting in the OrderID from the last step



Input Data Validation

- **Spatial data validation**
- **XML DTD Violation Handling**
 - The data for the TAG is ignored (current)
 - XML DTD validation (future)
- **Incorrect date format**
 - The data is ignored
- **Incorrect data type**
 - The collection/granule will be rejected
- **Keywords validation - GCMD (future)**
- **Source validation - GCMD (future)**
- **PSA name consolidation (future)**



Example 7 Submit input files for granule replacement with invalid spatial data

- **Goal:** To observe what is considered invalid spatial data
- **Steps:**
 - Take a look at ECHO for granules in a specific collection
 - Use 01_GetMOD01Granules.xml to query ECHO
 - Submit input xml files
 - FTP over 02_GranuleChanged_exp7.XML (Granule with invalid spatial data)
 - Wait for ingest summary report from ECHO
 - Granule with invalid spatial is not ingested
 - The first one covers an area more than half of earth in GEODETIC model
 - Take a look at ECHO again for granule in this specific collection
 - Use 03_GetMOD01Granules.xml: the granule is not updated



Example 8 Submit input files for collection deletion

- **Goal:** To observe what happens when provider submits a file for collection deletion
- **Step 1: Examine initial condition of collection**
 - Take a look at a specific collection MOD01 version 3
 - Use 01_GetMOD01Collection.xml to submit the query
 - 1 hit should be returned
- **Step 2: Examine initial condition of granules of that collection**
 - Take a look at ECHO for granules in a specific collection
 - Use 02_GetMOD01Granules.xml to query the system
 - You should see SC:MOD01.003:16578357, etc.
 - Record and save the CatalogItemid for the referenced granule
- **Step 3: Check that an order works for a granule in the collection**
 - Submit an order with the CatalogItemid saved from the last step (SC:MOD01.003:16578357)
 - Use 03_CrOrderSC_MOD01.003_16578357_exp8.xml to create an order
 - The order should create successfully
 - Record and save the OrderID
 - Clean up order with 04_CleanupOrder.xml



Example 8 Submit input files for collection deletion

- **Step 4: Ingest the collection deletion file**
 - Submit input xml files
 - FTP over 05_CollectionMOD01Del_exp8.XML (delete collection MOD01 version 3)
- **Step 5: Get notified of ingest completion**
 - Wait for ingest summary report from ECHO
 - Deletion information shown
- **Step 6: Check that collection was deleted**
 - Take a look at ECHO again for this collection
 - Use 06_GetMOD01Collection.xml to query ECHO for the collection
 - Should get 0 hits returned



Example 8 Submit input files for collection deletion

- **Step 7: Validate that granules are gone**
 - Take a look at ECHO again for a granule in this specific collection
 - Use 07_GetMOD01Granules.xml to submit a query for granules in the collection
 - Should get 0 hits
- **Step 8: Validate that orders are no longer possible**
 - Submit an order with SC:MOD01.003:16578357
 - Use 08_CrOrderSC_MOD01.003_16578357_exp8.xml to create an order
 - Paste the CatalogItemid that you saved from step 2 into the message
 - Catalog item deleted message should be returned
 - Record and save the OrderID
 - Use 09_CleanupOrder.xml and paste in the OrderID to clean up



Example 9 Submit input files with incorrect TAG

- **Goal:** To observe how ECHO currently handles incorrect XML input files
- **Steps:**
 - Submit input xml files
 - FTP over 01_IncorrectCollectionVersionID_exp9.XML (version ID TAG)
 - Wait for ingest error report from ECHO
 - <XMLValidationError>...</XMLValidationError>



Example 10 Submit input files with incorrect date format

- **Goal:** To observe how ECHO handles an incorrect date format
- The date format for the data provider is defined at initialization time and configured into the ECHO database
- **Steps:**
 - Submit input xml files
 - FTP over 01_IncorrectCollectionDateForm_exp10.XML (correct format: yyyy-mm-dd hh24:mi:ss)
 - Wait for ingest summary report from ECHO
 - New collection inserted
 - Take a look at ECHO for all collections
 - Using 02_GetTestCollectionsData.xml: the new collection has no temporal information



Example 11 Submit input files with incorrect data

- **Goal:** To observe how ECHO handles an incorrect data type
 - Wrong data type
 - Exceeds string maximum length
 - Required data
- **Steps:**
 - Submit input xml files (with bad data in basic information)
 - FTP over 01_IncorrectCollectionBadData_exp11.XML
 - Version ID: float 1.0.0.1
 - The collection with incorrect data is rejected
 - Summary report shows <Processed total="0">
 - Take a look at ECHO for all collections
 - Use 02_GetTestCollectionsData.xml to query ECHO for collections
 - The collection was not ingested



Error Recovery

- EOMG analyzes the error and communicates with data provider
- **Recovery Strategy**
 - Data provider makes correction on xml generation program to conform to DTDs and resubmit to ECHO for ingest
 - Define correct spatial resolution, refine spatial data, and resubmit to ECHO for ingest
 - Correct wrong data and resubmit to ECHO for ingest
 - If the error involves unique identifier then the old data should be removed from ECHO by either submit collection/granule deletion or request EOMG to work on it. Then resubmit correct data to ECHO for ingest
- **Data recovery via database recovery (rather difficult)**



Input Schedule and Pacing

- **Any time**
- **Approximately 10 minutes for every 1000 collections/granules**
 - Based on current hardware
 - Augmentation will be installed soon



Metadata Auditing

- **Request ad hoc report of your metadata in ECHO**
 - See sample report: ECHOgr_DEMO_PROVIDER0.rep
- **Monthly ingest volume report:**
 - http://www.v0ims.gsfc.nasa.gov/v0ims/DOCUMENTATION/ECHO/Granule_Collection.xls
- **Query ECHO via available web site such as EDG etc.**



Metadata Update Hands On Session



Update DTD

- <?xml version="1.0" encoding="UTF-8"?>
- <ELEMENT ProviderAccountService (UpdateMetadata)>
- <ELEMENT UpdateMetadata (Collection*, Granule*)>
- <ELEMENT Collection (Target+, (Add | Update | Delete)+)>
- <ELEMENT Granule (Target+, (Add | Update | Delete)+)>
- <ELEMENT Target (ID, ProviderLastUpdateDateTime, SaveDateTimeFlag?)>
- <ELEMENT Add (QualifiedTag, MetadataValue)>
- <ELEMENT Update (QualifiedTag, MetadataValue)>
- <ELEMENT Delete (QualifiedTag+)>
- <ELEMENT QualifiedTag (#PCDATA)>
- <ELEMENT MetadataValue (#PCDATA)>
- <ELEMENT ProviderLastUpdateDateTime (#PCDATA)>
- <ELEMENT SaveDateTimeFlag (SAVE | DONTSAVE)>
- <ELEMENT SAVE EMPTY>
- <ELEMENT ID (#PCDATA)>
- <ELEMENT DONTSAVE EMPTY>



FTP the input files

- FTP login user ID and Password
- Data directory
 - Data
 - collection
 - granule
 - browse
 - update
 - valids



Metadata Update Summary Report

- List of files processed
- Validation error
- Total collections/granules Processed
- Sample File



Getting Started

- Log in as DEMO_PROVIDER#
- Open up a dos window to [FTP://orval.gsfc.nasa.gov/DEMO_PROVIDER#/data/](ftp://orval.gsfc.nasa.gov/DEMO_PROVIDER#/data/)
 - Select the appropriate DEMO_PROVIDER# folder
 - Select data
 - You will use the update folder to place metadata update file
 - If you are watching the progress of ingest, you will need to refresh the FTP window
- Open up another browser window and point to <http://mail.yahoo.com> and log in as ECHOProvider#
 - This is where reports from ingest will be delivered



Example 1 Granule OnlineAccess URLs insertion

- Goal: To observe what happens when provider submits the Granule OnlineAccess URLs insertion xml
- Step1: put the 01_InsertOnlineAccessURL.xml in update folder
- Step2: Wait for ingest summary report from ECHO
 - Insertion information will be shown
 - Email will be sent to DEMO_PROVIDER# email account
 - Log into <http://mail.yahoo.com> with ECHOProvider# account



Example 1 Granule Online Resources insertion

- Goal: To observe what happens when provider submits the Granule Online Resources insertion xml
- Step1: put the 02_InsertOnlineResources.xml in update folder
- Step2: Wait for ingest summary report from ECHO
 - Insertion information will be shown
 - Email will be sent to DEMO_PROVIDER# email account
 - Log into <http://mail.yahoo.com> with ECHOProvider# account



Example 1 Query the granule

- Goal: query the granule to see the change of online access URL, online resource insertion.
- Step1: submit the 03_Query.xml



Example 1 clean up

- Submit 04_DeleteOnlineData.xml



Order Demonstration



Provider Order Management Service

- **POMS is used when a Provider needs to initiate communication with ECHO**
 - Typically, this is used for status updates
 - Also used for asynchronous replies and provider initiated order cancellations
- **Remember that a Provider Order is that piece of an order that pertains to a given Provider**
 - POMS knows which Provider is being referenced by who is logged in, so transactions refer to the OrderID and don't need to specify the complete ProviderOrderID
- **This service is crucial for providing the best customer experience**

Some Preparation

- **Several steps in the following slides will ask you to record and save an identifier from a result**
 - This is because certain identifiers are generated and we can't tell you what they will be in advance
 - It is best to select and copy the identifier right out of the browser
 - Save it in the Worksheet.txt file, or in XMLSpy
 - You will paste it into a message that the lab provides in the right position, and then submit the message

Asynchronous Vs. Synchronous Providers

- **ECHO allows for two types of response messages when it initiates a communication to the Provider**
 - **Synchronous:** The Provider will respond directly to the message.
 - It is assumed that this will happen in seconds
 - Responses include accept and reject
 - **Asynchronous:** The Provider cannot respond to the request immediately
 - Perhaps the calculation of the cost of an order will require looking up some information by hand
 - The Provider responds to the request with "I'll get back to you on that" and that completes the immediate response
 - ECHO will wait for an update to come through POMS
 - It is an operational task to check for aging unanswered requests

OrderIDs and TrackingIDs

- **ECHO uses the OrderID to uniquely identify an Order**
- **ECHO allows for Providers that have their own unique tracking IDs to be maintained by the system in addition to the OrderID**
 - This is called the ProviderTrackingID
 - It is initialized with the ECHO generated OrderID, but the Provider can change it
 - Provider initiated messages allow for both the OrderID and the ProviderTrackingID to identify the order being referenced
 - ECHO will check the ProviderTrackingID first
- **Clients will have access to both values so that they can check status with either ECHO or the Provider**

Example 1: Synchronous Acceptance of an Order

- **Demonstrate the scenario that a provider (DEMO_PROVIDER4 in the example) synchronously accepts the submit request for an order and decides to close the order immediately.**
 - Although this is one of six scenarios a SOAP provider handles submit requests, all ECS providers currently follows this scenario



Step 1: Create an Order

- **We start the scenario with order creation**
- **Create an order with an item belonging to DEMO_PROVIDER4**
 - File: 01_CreateOrder.xml
- **Record and save the order ID in the response, you are going to use this ID in the following steps.**



Step 2: Submit the Order

- **Set the user information for the order, validate, and submit**
- **File: 02_SetUserInformation.xml**
- **File: 03_ValidateOrder.xml**
- **File: 04_SubmitOrder.xml**



Step 3: Present the Order

- **Present the order**
 - File: 05_PresentOrder.xml (replace the order Id with the one you took down)
- **Verify that the user see the provider order in CLOSED state and provider's auto-responded timestamped message in the statusMessage field**
- **Verify that the order information such as total price, shipping/handling, etc. is provided**
- **Important!!!**
 - Record and save the provider tracking ID in the response, you are going to use this ID in the following steps.



Step 4: Close the Order

- **Log in as DemoUser4 and set provider context to DEMO_PROVIDER4**
- **Update status message**
 - provider can update status message to an order that is in any provider order state, even in a CLOSED state, although in this scenario provider is unlikely to do so
 - User name: DemoUser4
 - Password: Password101
 - File: 06_UpdateStatusMessage.xml (replace the provider tracking Id with the one you recorded)



Step 5: Present the Order

- **Log out from DemoUser4**
- **You will now be a guest again**
- **Present the order**
 - File: 07_PresentOrder.xml (replace the order Id with the one you recorded)
- **Verify that the statusMessage as a new entry**
 - It has a timestamp
 - It indicates that your order has shipped
- **End of Example 1**



Example 2: Asynchronous Acceptance (and immediate closure) of an Order

- Demonstrate the scenario that a provider (DEMO_PROVIDER1 in the example) synchronously accepts the submit request for an order and decides to close the order later using POMS transaction.



Step 1: Create an Order

- We start the scenario with order creation
- Create an order with an item belonging to DEMO_PROVIDER1
 - File: 01_CreateOrder.xml
- Record and save the order ID in the response, you are going to use this ID in the following steps.



Step 2: Submit the Order

- Set the user information for the order, validate, and submit
- File: 02_SetUserInformation.xml
- File: 03_ValidateOrder.xml
- File: 04_SubmitOrder.xml



Step 3: Present the Order

- Present the order
 - File: 05_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the user see the provider order in PROCESSING state and provider's auto-responded timestamped message in the statusMessage field
- Verify that order information is provided
- Record and save the provider tracking ID in the response, you are going to use this ID in the following steps.



Step 4: Close the Order

- Switch user: Log in as DemoUser1 and set provider context to DEMO_PROVIDER1
- Close provider order
 - User name: DemoUser1
 - Password: Password101
 - File: 06_CloseProviderOrder.xml (replace the provider tracking Id with the one you took down)



Step 5: Present the Order

- Log out from DemoUser1
- Present the order
 - File: 07_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the provider order state changed to CLOSED, the updated message in the last step is appended and timestamped in the statusMessage field
- End of Example 2



Example 3: Asynchronous Acceptance (without closure) of an Order

- Demonstrate the scenario that a provider (DEMO_PROVIDER3 in the example) uses POMS transaction to asynchronously accept the submit request for an order and decides to close the order immediately.



Step 1: Create an Order

- We start the scenario with order creation
- Create an order with an item belonging to DEMO_PROVIDER3
 - File: 01_CreateOrder.xml
- Record and save the order ID in the response, you are going to use this ID in the following steps.



Step 2: Submit the Order

- Set the user information for the order, validate, and submit
- File: 02_SetUserInformation.xml
- File: 03_ValidateOrder.xml
- File: 04_SubmitOrder.xml



Step 3: Present the Order

- Present the order
 - File: 05_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the user see the provider order in SUBMITTING state and provider's auto-responded timestamped message in the statusMessage field
- Verify that no order information provided
- Record and save the provider tracking ID in the response, you are going to use this ID in the following steps.



Step 4: Close the Order

- Log in as DemoUser3 and set provider context to DEMO_PROVIDER3
- Accept the provider order (specify NONE for UpdateMechanism)
 - User name: DemoUser3
 - Password: Password101
 - File: 06_AcceptProviderOrderSubmission.xml (replace the provider tracking Id with the one you took down)



Step 5: Present the Order

- Log out from DemoUser3
- Present the order
 - File: 07_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the provider order state changed to CLOSED, the updated message in the last step is appended and timestamped in the statusMessage field
- Verify that the order information is provided
- End of Example 3



Example 4: Asynchronous Acceptance (and eventual closure) of an Order

- Demonstrate the scenario that a provider (DEMO_PROVIDER3 in the example) uses POMS transaction to asynchronously accept the submit request for an order and decides to close the order later.



Step 1: Create an Order

- We start the scenario with order creation
- Create an order with an item belonging to DEMO_PROVIDER3
 - File: 01_CreateOrder.xml
- Record and save the order ID in the response, you are going to use this ID in the following steps.



Step 2: Submit the Order

- Set the user information for the order, validate, and submit
- File: 02_SetUserInformation.xml
- File: 03_ValidateOrder.xml
- File: 04_SubmitOrder.xml



Step 3: Present the Order

- Present the order
 - File: 05_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the user see the provider order in SUBMITTING state and provider's auto-responded timestamped message in the statusMessage field
- Verify that no order information is provided
- Record and save the provider tracking ID in the response, you are going to use this ID in the following steps.



Step 4: Accept the Order

- Switch user: Log in as DemoUser3 and set provider context to DEMO_PROVIDER3
- Accept the provider order (specify MANUAL for UpdateMechanism)
 - User name: DemoUser3
 - Password: Password101
 - File: 06_AcceptProviderOrderSubmission.xml (replace the provider tracking Id with the one you took down)



Step 5: Present the Order

- Log out from DemoUser3
- Present the order
 - File: 07_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the provider order state changed to PROCESSING, the updated message in the last step is appended and timestamped in the statusMessage field
- Verify that the order information is provided



Step 6: Close the Order

- Log in as DemoUser3 and set provider context to DEMO_PROVIDER3
- Close provider order
 - User name: DemoUser3
 - Password: Password101
 - File: 08_CloseProviderOrder.xml (replace the provider tracking Id with the one you took down)



Step 7: Present the Order

- Log out from DemoUser3
- Present the order
 - File: 09_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the provider order state changed to CLOSED, the updated message in the last step is appended and timestamped in the statusMessage field
- End of Example 4



Example 5: Synchronous Rejection of an Order

- Demonstrate the scenario that a provider (DEMO_PROVIDER2 in the example) synchronously rejects the submit request for an order.



Step 1: Create an Order

- We start the scenario with order creation
- Create an order with an item belonging to DEMO_PROVIDER2
 - File: 01_CreateOrder.xml
- Record and save the order ID in the response, you are going to use this ID in the following steps.



Step 2: Submit the Order

- Set the user information for the order, validate, and submit
- File: 02_SetUserInformation.xml
- File: 03_ValidateOrder.xml
- File: 04_SubmitOrder.xml



Step 3: Present the Order

- Present the order
 - File: 05_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the user see the provider order in SUBMIT_REJECTED state and provider's auto-responded timestamped message in the statusMessage field
- End of Example 5



Example 6: Asynchronous Rejection of an Order

- Demonstrate the scenario that a provider (DEMO_PROVIDER3 in the example) uses POMS transaction to asynchronously reject the submit request for an order.



Step 1: Create an Order

- We start the scenario with order creation
- Create an order with an item belonging to DEMO_PROVIDER3
 - File: 01_CreateOrder.xml
- Record and save the order ID in the response, you are going to use this ID in the following steps.



Step 2: Submit the Order

- Set the user information for the order, validate, and submit
- File: 02_SetUserInformation.xml
- File: 03_ValidateOrder.xml
- File: 04_SubmitOrder.xml



Step 3: Present the Order

- Present the order
 - File: 05_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the user see the provider order in SUBMITTING state and provider's auto-responded timestamped message in the statusMessage field
- Record and save the provider tracking ID in the response, you are going to use this ID in the following steps.



Step 4: Reject the Order

- Log in as DemoUser3 and set provider context to DEMO_PROVIDER3
- Reject the provider order
 - User name: DemoUser3
 - Password: Password101
 - File: 06_RejectProviderOrderSubmission.xml (replace the provider tracking Id with the one you took down)



Step 5: Present the Order

- Log out from DemoUser3
- Present the order
 - File: 07_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the provider order state changed to SUBMIT_REJECTED, the updated message in the last step is appended and timestamped in the statusMessage field
- End of Example 6



Example 7: Synchronous Quoting of an Order

- Demonstrate the scenario that a provider (DEMO_PROVIDER1 in the example) synchronously accepts the quote request for an order and supplies the provider quote.
 - There are four scenarios that a SOAP provider handles quote request

Step 1: Create an Order

- We start the scenario with order creation
- Create an order with an item belonging to DEMO_PROVIDER1
 - File: 01_CreateOrder.xml
- Record and save the order ID in the response, you are going to use this ID in the following steps.

Step 2: Quote the Order

- Set the user information for the order, validate, and submit
- File: 02_SetUserInformation.xml
- File: 03_ValidateOrder.xml
- File: 04_QuoteOrder.xml

Step 3: Present the Order

- Present the order
 - File: 05_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the user see the provider order in QUOTED state and that quote is provided
- End of Example 7

Example 8: Asynchronous Quoting of an Order

- Demonstrate the scenario that a provider (DEMO_PROVIDER3 in the example) uses POMS transaction to asynchronously accept the quote request for an order.
- Note: There is a known issue that the SupplyQuote transaction doesn't not work which is fixed in version 6.0.

Step 1: Create an Order

- We start the scenario with order creation
- Create an order with an item belonging to DEMO_PROVIDER3
 - File: 01_CreateOrder.xml
- Record and save the order ID in the response, you are going to use this ID in the following steps.

Step 2: Quote the Order

- Set the user information for the order, validate, and submit
- File: 02_SetUserInformation.xml
- File: 03_ValidateOrder.xml
- File: 04_QuoteOrder.xml



Step 3: Present the Order

- Present the order
 - File: 05_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the user see the provider order in **QUOTING** state and no quote is provided yet
- Record and save the provider tracking ID in the response, you are going to use this ID in the following steps.



Step 4: Quote the Order (as a Provider)

- Log in as DemoUser3 and set provider context to **DEMO_PROVIDER3**
- Supply provider quote
 - User name: DemoUser3
 - Password: Password101
 - File: 06_SupplyProviderQuote.xml (replace the provider tracking Id with the one you took down)



Step 5: Present the Order

- Log out from DemoUser3
- Present the order
 - File: 07_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the provider order state changed to **QUOTED**, and that quote is provided
- End of Example 8



Example 9: Synchronous Rejection of a Quoting Request

- Demonstrate the scenario that a provider (**DEMO_PROVIDER2** in the example) synchronously rejects the quote request for an order.



Step 1: Create an Order

- We start the scenario with order creation
- Create an order with an item belonging to **DEMO_PROVIDER2**
 - File: 01_CreateOrder.xml
- Record and save the order ID in the response, you are going to use this ID in the following steps.



Step 2: Quote the Order

- Set the user information for the order, validate, and submit
- File: 02_SetUserInformation.xml
- File: 03_ValidateOrder.xml
- File: 04_QuoteOrder.xml



Step 3: Present the Order

- Present the order
 - File: 05_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the user see the provider order in QUOTE_REJECTED state



Step 4: Re-Quote the Order

- Quote the order again
 - File: 06_QuoteOrder.xml (replace the order Id with the one you took down)
- Verify that an error message in the response not allowing to quote an order that has been rejected
- Note: the error message is not appropriate, this is a known issue.
- End of Example 9



Example 10: Asynchronous Rejection of a Quoting Request

- Demonstrate the scenario that a provider (DEMO_PROVIDER3 in the example) uses POMS transaction to asynchronously reject the quote request for an order.



Step 1: Create an Order

- We start the scenario with order creation
- Create an order with an item belonging to DEMO_PROVIDER3
 - File: 01_CreateOrder.xml
- Record and save the order ID in the response, you are going to use this ID in the following steps.



Step 2: Quote the Order

- Set the user information for the order, validate, and submit
- File: 02_SetUserInformation.xml
- File: 03_ValidateOrder.xml
- File: 04_QuoteOrder.xml



Step 3: Present the Order

- **Present the order**
 - File: 05_PresentOrder.xml (replace the order Id with the one you took down)
- **Verify that the user see the provider order in QUOTING state and no quote is provided yet.**
- **Record and save the provider tracking ID in the response, you are going to use this ID in the following steps.**



Step 4: Reject the Quote (as a Provider)

- **Log in as DemoUser3 and set provider context to DEMO_PROVIDER3**
- **Reject the provider order**
 - User name: DemoUser3
 - Password: Password101
- File: 06_RejectProviderQuote.xml (replace the provider tracking Id with the one you took down)



Step 5: Present the Order

- **Log out from DemoUser3**
- **Present the order**
 - File: 07_PresentOrder.xml (replace the order Id with the one you took down)
- **Verify that the provider order state changed to QUOTE_REJECTED**
- **End of Example 10**



Example 11: Synchronous Cancellation of an Order

- **Demonstrate the scenario that a provider (DEMO_PROVIDER1 in the example) synchronously accepts the cancel request for an order.**
 - There are four scenarios that a SOAP provider handles cancel request
 - Only registered user of ECHO can cancel an order



Step 1: Login and Create an Order

- **Log in as the user**
 - User name: DemoUser#
 - Password: Password101
- **Create an order with an item belonging to DEMO_PROVIDER1**
 - File: 01_CreateOrder.xml
- **Record and save the order ID in the response, you are going to use this ID in the following steps.**



Step 2: Submit the Order

- **Set the user information for the order, validate, and submit**
- **File: 02_SetUserInformation.xml**
- **File: 03_ValidateOrder.xml**
- **File: 04_SubmitOrder.xml**



Step 3: Present the Order

- **Present the order**
 - File: 05_PresentOrder.xml (replace the order Id with the one you took down)
- **Verify that the user see the provider order in PROCESSING state**



Step 4: Cancel the Order

- **Cancel the order**
 - File: 06_CancelOrder.xml (replace the order Id with the one you took down)
- **Note: user can only cancel an order in PROCESSING state**



Step 5: Present the Order

- **Present the order**
 - File: 07_PresentOrder.xml (replace the order Id with the one you took down)
- **Verify that the user see the provider order in CANCELLED state**
- **Log out from user DemoUser#**
- **End of Example 11**



Example 12: Asynchronous Cancellation of an Order

- **Demonstrate the scenario that a provider (DEMO_PROVIDER3 in the example) uses POMS transaction to asynchronously accept the cancel request for an order.**



Step 1: Login and Create an Order

- **We start the scenario with order creation**
- **Log in as the user**
 - User name: DemoUser#
 - Password: Password101
- **Create an order with an item belonging to DEMO_PROVIDER3**
 - File: 01_CreateOrder.xml
- **Record and save the order ID in the response, you are going to use this ID in the following steps.**



Step 2: Submit the Order

- **Set the user information for the order, validate, and submit**
- **File: 02_SetUserInformation.xml**
- **File: 03_ValidateOrder.xml**
- **File: 04_SubmitOrder.xml**



Step 3: Present the Order

- **Present the order**
 - File: 05_PresentOrder.xml (replace the order Id with the one you took down)
- **Verify that the user see the provider order in SUBMITTING state**
- **Record and save the provider tracking ID in the response, you are going to use this ID in the following steps.**



Step 4: Accept the Order

- **Log in as DemoUser3 and set provider context to DEMO_PROVIDER3**
- **Accept the submitted order (specify MANUAL in UpdateMechanism, user can only cancel an order in PROCESSING state, this is the setup for cancellation)**
 - User name: DemoUser3
 - Password: Password101
- File: 06_AcceptProviderOrderSubmission.xml (replace the provider tracking Id with the one you took down)



Step 5: Cancel the Order

- **Log out from DemoUser3 and set provider context to DEMO_PROVIDER3**
- **Login to DemoUser#**
- **Cancel the order**
 - File: 07_CancelOrder.xml (replace the order Id with the one you took down)



Step 6: Present the Order

- **Present the order**
 - File: 08_PresentOrder.xml (replace the order Id with the one you took down)
- **Verify that the provider order state changed to CANCELLING**



Step 7: Cancel the Order (as the Provider)

- **Log in as DemoUser3 and set provider context to DEMO_PROVIDER3**
- **Cancel provider order**
 - User name: DemoUser3
 - Password: Password101
 - File: 09_CancelProviderOrder.xml (replace the provider tracking Id with the one you took down)



Step 8: Present the Order

- **Log out from DemoUser3**
- **Login as DemoUser#**
- **Present the order**
 - File: 10_PresentOrder.xml (replace the order Id with the one you took down)
- **Verify that the provider order state changed to CANCELLED**
- **Log out from DemoUser#**
- **End of Example 12**



Example 13: Asynchronous Rejection of a Cancellation Request

- Demonstrate the scenario that a provider (DEMO_PROVIDER3 in the example) asynchronously rejects the cancel request for an order.



Step 1: Login and Create an Order

- We start the scenario with order creation
- Log in as the user
 - User name: DemoUser#
 - Password: Password101
- Create an order with an item belonging to DEMO_PROVIDER3
 - File: 01_CreateOrder.xml
- Record and save the order ID in the response, you are going to use this ID in the following steps.



Step 2: Submit the Order

- Set the user information for the order, validate, and submit
- File: 02_SetUserInformation.xml
- File: 03_ValidateOrder.xml
- File: 04_SubmitOrder.xml



Step 3: Present the Order

- Present the order
 - File: 05_PresentOrder.xml (replace the order Id with the one you took down)
- Verify that the user see the provider order in SUBMITTING state
- Record and save the provider tracking ID in the response, you are going to use this ID in the following steps.



Step 4: Accept the Order

- Logout from DemoUser#
- Log in as DemoUser3 and set provider context to DEMO_PROVIDER3
- Accept the submitted order (specify MANUAL in UpdateMechanism, user can only cancel an order in PROCESSING state, this is the setup for cancellation)
 - User name: DemoUser3
 - Password: Password101
 - File: 06_AcceptProviderOrderSubmission.xml (replace the provider tracking Id with the one you took down)



Step 5: Cancel the Order

- Log out from DemoUser3
- Login as DemoUser#
- Cancel the order
 - File: 07_CancelOrder.xml (replace the order Id with the one you took down)



Step 6: Present the Order

- **Present the order**
 - File: 08_PresentOrder.xml (replace the order Id with the one you took down)
- **Verify that the provider order state changed to CANCELLING**



Step 7: Reject the Cancellation Request (as the Provider)

- **Logout from DemoUser#**
- **Log in as DemoUser3 and set provider context to DEMO_PROVIDER3**
- **Reject the cancel request**
 - User name: DemoUser3
 - Password: Password101
 - File: 09_RejectProviderOrderCancellation.xml (replace the provider tracking Id with the one you took down)



Step 8: Present the Order

- **Log out from DemoUser3**
- **Login as DemoUser#**
- **Present the order**
 - File: 10_PresentOrder.xml (replace the order Id with the one you took down)
- **Verify that the provider order state remains PROCESSING, and status message gives reason for rejection**



Step 9: Try to Cancel Again

- **Cancel the order again**
 - File: 11_CancelOrder.xml (replace the order Id with the one you took down)
- **Verify that an error message in the response not allowing to cancel an order that has been rejected**
- **Log out from DemoUser#**
- **End of Example 13**



Data Management Service



A Quick Summary from Yesterday

- **Why do we have the DataManagementService?**
 - Gives providers the ability to manage access control lists
- **What are the key components of access control lists?**
 - Groups (aggregations of users)
 - Conditions (Phraselets)
 - Rules (Restrictions or Permissions)



Typical Scenarios

- **A provider sends ECHO a new collection**
 - The provider does not want the entire ECHO community to be able to view or order the collection until they have verified its accuracy.
 - The provider wishes to allow access to the collection to a select group of test users.
- **An instrument malfunctions**
 - The provider wishes to prevent everyone from viewing and ordering data during the time period when 'bad' data were received
- **In either case, the provider knows:**
 - What collection (or granule) they wish to control
 - How they wish to control it

Restrictions & Permissions

- **Restrictions prevent access to metadata and apply to the entire ECHO community**
- **Permissions grant access to metadata and apply to a specific group of users**
- **Evaluation is "Optimistic"**
 - If a user belongs to a group to which a permission has been granted, the user is given access

Our Hands-On Session

- **Goals of our Session**
 - Gain a solid understanding of the access control list concepts (groups, conditions, rules) as well as how to apply them to restrict and grant access to metadata
 - Familiarize you with the API used to manage your access control lists
- **Measures of Success**
 - By the end of the session, you will be able to
 - Manage your own access control lists
 - Understand how groups, conditions, and rules work together
 - Understand how restrictions and permissions are evaluated

Example #1: Restricting A Bad Granule

- **The Set-Up:**
 - The provider 'DEMO_PROVIDER#' exists
 - A specific granule was not ingested properly and the provider wishes to restrict all access to it indefinitely.
- **Assumption**
 - Set provider context to DEMO_PROVIDER#

The Plan

- **Issue a query to confirm the problem**
- **Create a Boolean 'True' Condition**
- **Create a Restriction**
 - Using the Boolean 'True' Condition
 - Referencing the 'Demonstration Granule'
- **Clean-Up**

Issue a Query

- **Look specifically for the 'Demonstration Granule'**
- **You should receive the granule metadata**

- **File: 01_Query.xml**

Create a Boolean 'True' Condition

- Not much to look at here
- Name: true condition
- BooleanFlag: True

- File: 02_CreateBooleanCondition.xml



Create a Restriction

- ActionType: View
 - We would also create a Restriction for Order
- ConditionType: Boolean
- ConditionName: true condition
- Comparator: Equals
- DataType: Granule
- DataValue: 'Demonstration Granule'

- File: 03_CreateRestriction.xml



Side Topic: Comparators and Conditions

- In our example, we have created a Boolean Condition with a BooleanFlag of 'true'.
- The Comparator, which exists at the rule level, tells ECHO how to evaluate the Condition.
- The flag is the Boolean Condition is only 1/2 of the equation - the Comparator is the other half.
 - 'true' is meaningless
 - 'Equals' 'true' provides an evaluation mechanism
 - 'Not Equals' 'false' the same as 'Equals' 'true'
- This will become more intuitive in later examples



Re-Issue the Query

- You should receive an XML message indicating that the granule is not visible.

- File: 04_Query.xml

- (Congratulations - You have just created your first access control list!!)



List your Rules and Conditions

- Now that you have a restriction and a condition, let's look at how to manage them
 - ListRules shows the name of all restrictions and permissions you have created
 - ListConditions shows the name of all conditions (and their type) you have created
- Presentation of Rules and Conditions displays all of the attributes of the rule or condition
 - Listing just displays the name of the rule or condition

- File: 05_ListRules.xml
- File: 06_ListConditions.xml



Presenting your Rules and Conditions

- Not much to really explain here

- File: 07_PresentRule.xml
- File: 08_PresentBooleanCondition.xml



Clean-Up

- Delete your condition
- This will fail because a rule references it.

- File: 09_DeleteCondition.xml



Clean-Up

- Solution: Delete your rule, and then delete your condition

- File: 10_DeleteRule.xml
- File: 11_DeleteCondition.xml



Confirmation

- Re-Issue your Query
- Demonstration Granule should be visible

- File: 12_Query.xml



Summary

- We located a granule we wanted to restrict
- We created a boolean true condition
- We created a restriction that uses the boolean true condition to prevent viewing access to the granule
- We cleaned up



Example #2: Restricting Lots of Bad Granules

- The Set-Up:
 - The provider 'DEMO_PROVIDER#' exists
 - All of the granules within the two months of 2001 were not properly ingested.
 - The provider wishes to restrict viewing access to the bad granules



The Plan

- Issue a query to confirm the problem
- Create a Temporal Condition
- Create a Restriction
 - Using the Temporal Condition
 - Referencing all granules



Issue a Query

- Look for granules
- Start: Jan 1, 2002
- Stop: Dec 29, 2002
- You should receive several granules

- File: 01_Query.xml



Create a Temporal Condition

- Name: early 2002
- Start Date: Jan 1, 2002
- Stop Date: Dec 29, 2002

- File: 02_CreateTemporalCondition.xml



Create a Restriction

- ActionType: View
 - We would also create a Restriction for Order
- ConditionType: Temporal Condition
- ConditionName: early 2002
- Comparator: Equals
- DataType: Granule
- DataValue: ALL (keyword)

- File: 03_CreateRestriction.xml



Comparators and Conditions

- In our example, we have created a Temporal Condition
- The Comparator explains how to evaluate the Start and Stop Date.
 - An Equals comparator instructs ECHO to look at the dates between the Start and Stop Date.
 - A Not Equals comparator instructs ECHO to look at the dates outside the Start and Stop Date.



Re-Issue the Query

- You should receive an XML message indicating that some granules were hidden, and others were not.
 - The query covers Jan 1 2002 - Dec 29 2002
 - The granules between Jan 1 2002 and Feb 29 2002 are restricted
 - The granules between March 1 2002 - Dec 29 2002 are visible

- File: 04_Query.xml



Presenting your Rules and Conditions

- The condition presentation includes the Start and Stop Date
- The rule presentation includes all other facts:
 - Type of Access
 - Type of condition referenced
 - Name of condition referenced
 - Comparator used

File: 05_PresentRule.xml

File: 06_PresentTemporalCondition.xml



Clean-Up

- Delete your rule
- Delete your condition

- File: 07_DeleteRule.xml
- File: 08_DeleteCondition.xml



Confirmation

- Re-Issue your Query
- The previous hidden granules are now visible

- File: 09_Query.xml



Summary

- We located a time frame during which period bad granule metadata was loaded into ECHO.
- We created a temporal condition that encompassed the time frame.
- We created a restriction using the temporal condition to prevent access to all granules in the time frame.



Example #3: Restricting 'young' metadata

- The Set-Up:
 - A provider is uncomfortable ingesting metadata into ECHO unless the data is hidden and users are unable to view it and order it
 - This is perceived to be the common case



The Plan

- Issue a query to confirm that 'young data' is visible.
- Create a Rolling Temporal Condition
- Create a Restriction
 - Using the Rolling Temporal Condition
 - Referencing all granules



Issue a Query

- Look for granules created within the last 12 months.
- Start: Sept 1, 2002
- Stop: Dec 29, 2002
- You should receive several granules

- File: 01_Query.xml



Create a Rolling Temporal Condition

- **Name:** young data condition
- **Duration:** 12 months

- **File:** 02_CreateRollingTemporalCondition.xml



Create a Restriction

- **ActionType:** View
 - We would also create a Restriction for Order
- **ConditionType:** Rolling Temporal Condition
- **ConditionName:** young data condition
- **Comparator:** Less Than
- **DataType:** Granule
- **DataValue:** ALL (keyword)

- **File:** 03_CreateRestriction.xml



Conditions and Comparators

- In our example, we have created a Rolling Temporal Condition
- The Comparator explains how to evaluate the Duration.
 - A Less Than comparator instructs ECHO to look at dates between now and N days ago, where N is the duration specified in the Condition.
 - A Greater Than comparator instructs ECHO to look at dates beyond N days ago.



Re-Issue the Query

- You should receive an XML message indicating that some granules were hidden, and others were not.
 - The query covers Sept 1 2002 - Dec 29 2002
 - The granules less than 12 months old are hidden
 - The granules older than 12 months old are visible

- **File:** 04_Query.xml



Presenting Your Rules and Conditions

- The condition presentation includes Duration and Temporal Units.
- The rule presentation includes all other facts:
 - Type of Access
 - Type of condition referenced
 - Name of condition referenced
 - Comparator used

File: 05_PresentRule.xml

File: 06_PresentRollingTemporalCondition.xml



Clean-Up

- Delete your rule
- Delete your condition

- **File:** 07_DeleteRule.xml
- **File:** 08_DeleteCondition.xml



Confirmation

- Re-Issue your Query
- The previous hidden granules are now visible

- File: 09_Query.xml



Summary

- We wanted to block metadata less than 12 months old
- We created a rolling temporal condition with a duration of 12 months
- We created a restriction against all granules and referenced the 12 months rolling temporal condition



Example #4: Restricting Granules Within a Collection

- **The Set-Up:**
 - We have identified a subset of granules within a Collection that were not properly ingested.
 - Said granules exist within a specific time frame
 - We want to restrict those granules from being viewed and ordered
- **What is different?**
 - We previously were working only with granules.
 - Now we're going to use the collection as a mechanism for partially identifying the granules to restrict.



The Plan

- Issue a query to confirm that all the data is visible
- Create a Temporal Condition
- Create a Restriction
 - Using the Temporal Condition
 - Referencing a specific collection



Issue a Query

- Look for granules
- Start: Jan 1, 2002
- Stop: Dec 31, 2002
- We don't care about the Collection the granules belong to

- File: 01_Query.xml



Create a Temporal Condition

- Name: year 2002
- Start Date: Jan 1, 2002
- Stop Date: Dec 31, 2002

- File: 02_CreateTemporalCondition.xml



Create a Restriction

- **ActionType: View**
 - We would also create a Restriction for Order
- **ConditionType: Temporal Condition**
- **ConditionName: year 2002**
- **Comparator: Equals**
- **DataType: Collection**
- **DataValue: Equally distributed insert time V001**

- **File: 03_CreateRestriction.xml**

Re-Issue the Query

- **You should receive an XML message indicating that some granules were hidden, and others were not.**
 - The query covers Jan 1 2002 - Dec 31 2002
 - The granules between Jan 1 2002 - Dec 31 2002 belonging to the restricted collection are not visible
 - The granules between Jan 1 2002 - Dec 31 2002 not belonging to the restricted collection are visible

- **File: 04_Query.xml**

Presenting Your Rules and Conditions

- **The condition presentation includes a Start Date and Stop Date**
- **The rule presentation includes all other facts:**
 - Type of Access
 - Type of condition referenced
 - Name of condition referenced
 - Comparator used

File: 05_PresentRule.xml

File: 06_PresentTemporalCondition.xml

Issue a Different Query

- **Look for the Collection 'Equally distributed insert time V001'**
- **You will receive results even though you created a restriction against the 'Equally distributed insert time V001' collection.**
- **Why?**

- **File: 07_Query.xml**

Clean-Up

- **Before we answer that, let's clean-up**
- **Delete your rule**
- **Delete your condition**

- **File: 08_DeleteRule.xml**
- **File: 09_DeleteCondition.xml**

Re-Issue the Initial Query

- **You should receive an XML message with lots of results**
 - No granules should be hidden

- **File: 10_Query.xml**

Side Topic: What Gets Restricted?

- In the previous examples, granule level metadata was restricted.
- In order to restrict collection level metadata, you must use a boolean condition with a collection DataType.



Example #5: Restricting Collection MetaData

- **The Set-Up:**
 - We have identified the metadata of a particular collection to be bad
 - We wish to prevent all users from viewing the collection's metadata as well as the granules belonging to the collection



The Plan

- Issue a query to confirm that the collection is visible
- Create a Boolean 'True' Condition
 - Uses the Boolean 'True' Condition
 - References the 'Equally distributed insert time V001' collection
- Create a Restriction
- Clean-Up



Issue a Query

- Look for collections
- With the name 'Equally distributed insert time V001'
- You will receive results

- File: 01_Query.xml



Create a Boolean 'True' Condition

- Name: true condition
- BooleanFlag: True

- File: 02_CreateBooleanCondition.xml



Create a Restriction

- ActionType: View
 - We would also create a Restriction for Order
- ConditionType: Boolean
- ConditionName: true condition
- Comparator: Equals
- DataType: Collection
 - We previously used a boolean condition against a Granule
- DataValue: 'Equally distributed insert time V001'

- File: 03_CreateRestriction.xml



Re-Issue the Query

- Look for collections
- With the name 'Equally distributed insert time V001'
- You will not find any results

- File: 04_Query.xml



Issue a Query for Granules

- Look for Granules
- Belonging to the 'Equally distributed insert time V001' collection
- You will not find any results
- All metadata related to the 'Equally distributed insert time V001' collection has been hidden

- File: 05_Query.xml



Clean-Up

- Delete your rule
- Delete your condition

- File: 06_DeleteRule.xml
- File: 07_DeleteCondition.xml



Re-Issue the Query

- You will receive results

- File: 08_Query.xml
- File: 09_Query.xml



Summary

- We identified a collection as 'bad'
- We created a temporal condition during a time frame where bad granules were ingested
- We created a restriction to prevent viewing of the 'bad' granules.
- We then realized that temporal conditions do not prevent access to collection level metadata.
 - We deleted our previous restriction and collection
 - We created a boolean condition, and used it to restrict access to a collection
 - We then found that collection level metadata and granule metadata were restricted



Example #6: Restricting All MetaData

- The Set-Up:
 - Something horrible has happened
 - You wish to prevent access to all of your metadata holdings within ECHO.



The Plan

- Issue a query to confirm that data exists
- Create a 'true' boolean condition
- Use the 'true' boolean condition to restrict access to all collections



Issue a Query

- Look for all collections
- You will find results

- File: 01_Query.xml



Create a Boolean 'true' Condition

- Name: true condition
- BooleanFlag: True

- File: 02_CreateBooleanCondition.xml



Create a Restriction

- ActionType: View
 - We would also create a Restriction for Order
- ConditionType: Boolean
- ConditionName: true condition
- Comparator: Equals
- DataType: Collection
- DataValue: 'ALL' (keyword)

- File: 03_CreateRestriction.xml



Re-Issue the Query

- Look for all collections
- You won't find any results

- File: 04_Query.xml



Issue Another Query

- Look for all granules
- You won't find any results

- File: 05_Query.xml



Clean-Up

- Delete your rule
- Delete your condition

- File: 06_DeleteRule.xml
- File: 07_DeleteCondition.xml



Re-Issue the Queries

- Look for all granules
- Look for all collections
- You will find several results

- File: 08_Query.xml
- File: 09_Query.xml



What Have We Learned So Far?

- **Restrictions**
 - How to restrict a specific granule.
 - How to restrict all granules
 - How to restrict granules within a specific time range
 - How to restrict granules within the last N days
 - How to restrict a specific collection
 - How to restrict all collections
- **So how do we grant access?**



Permissions

- **Follow the same pattern as restrictions**
 - Reference a condition and a comparator to provide an evaluation mechanism
 - Reference a data item (a collection or a granule)
 - Can alternatively use the 'ALL' keyword
- **Additional Field**
 - Group
 - Allows you to specify the specific users you want to grant access to



Group Primer

- **Groups are created by a provider**
 - The creator becomes the 'owner' of the group
- **The group is governed by the management team.**
 - Group Managers have full control over the group
 - They can add or remove other managers
 - They can add or remove members
- **The group's membership roster**
 - Consists of ECHO users
 - Not public information (the membership list cannot be displayed except by a manager of the group)



How Groups Relate to Permissions

- Providers can create groups and delegate management of the group to other ECHO users
- The managers of the group can add ECHO users to the group's membership roster
- The provider can then grant a permission to the group, thus giving a subset of the ECHO community special permission to view their metadata holdings that would otherwise be restricted.



Example #7: Creating a Group

- **The Set-Up:**
 - We have identified a subset of the ECHO community that we wish to give permission on our metadata holdings to
 - We will later on create a permission that uses the group we create here



The Plan

- **Create a Group**
- **Add a Member to the Group**
- **Present the Group**
- **List the Group's Members**



Create a Group

- **Name: Group#**
- **Managers: Provider#**

- **File: 01_CreateGroup.xml**



Add a Member to the Group

- **Group Name: Group#**
- **Member's UserName: DemoUser#**
- **Notify: true**
 - When the user is added to the group, a confirmation email is sent to the user's informing them of their membership in the group.

- **File: 02_AddMember.xml**



Present the Group

- Shows the group name, description, and management team.
- Does not show the membership roster

- **File: 03_PresentGroupInformation.xml**



List the Group's Members

- This transaction can only be called by a group manager.
- Since the CreateGroup message you submitted listed you as a manager you are able to execute this transaction

- **File: 04_ListMembers.xml**



Clean-Up

- Destroy your group
- File: 05_DestroyGroup.xml



Summary

- We've just
 - Create a Group
 - Added a member to the Group
 - Presented the group's information
 - Listed the group's members
- Similar transactions exist for the management team:
 - ListManagers
 - UpdateGroupInformation
- Group Name's are unique across the entire ECHO system



Example #8: Granting a Permission on a Granule

- The Set-Up:
 - A provider has identified a particular granule as being 'bad'
 - The provider wishes to restrict the entire ECHO community from viewing the 'bad' granule
 - The provider wishes to allow their internal test team to view the granule
 - Perhaps they wish to monitor the granule until they perform another ingest to correct it
 - When the granule is 'fixed', they will remove the restriction



The Plan

- Perform a Query to confirm the 'bad' granule is visible
- Create a Restriction to prevent the entire ECHO community from viewing the 'bad' granule.
- Re-Issue the query to confirm the 'bad' granule is hidden
- Create a Group and add a test user to the group.
- Assign permission to the group to view the 'bad' granule.
- Re-Issue the initial query (as a member of the group) to confirm that the 'bad' granule is visible.



Issue a Query

- Look specifically for the 'Equally distributed insert time: granule #0: 2000-01-01 00:00:00.000'
- You will receive a hit
- File: 01_Query.xml



Restrict the Granule

- Create a Boolean Condition
- Restrict the granule using the boolean condition
- File: 02_CreateBooleanCondition.xml
- File: 03_CreateRestriction.xml



Re-Issue the Query

- Look specifically for the 'Equally distributed insert time: granule #0: 2000-01-01 00:00:00.000'
- You will not receive any hits
- File: 04_Query.xml



Create a Group

- Name: Group#
- Manager: DemoUser#
- File: 05_CreateGroup.xml



Add a Member

- Group#
- DemoUser#
 - We're adding our self as a member to the group
 - Typically we would add a test user instead of our self
- File: 06_AddMember.xml



Create the Permission

- ActionType: View
- ConditionType: Boolean
- ConditionName: true condition
- Comparator: Equals
- Group: Group#
- DataType: Granule
- DataValue: 'Equally distributed insert time: granule #0: 2000-01-01 00:00:00.000'
- File: 07_CreatePermission.xml



Re-Issue the Query

- Look specifically for the 'Equally distributed insert time: granule #0: 2000-01-01 00:00:00.000'
- You'll receive a hit
 - This is because you are a member of a group that has explicit permission to view the 'Equally distributed insert time: granule #0: 2000-01-01 00:00:00.000'
 - If you were another user of ECHO, you would not be able to view the 'Equally distributed insert time: granule #0: 2000-01-01 00:00:00.000'
 - An exercise for home
 - Use another account to run the query and examine the results
- File: 08_Query.xml



Clean-Up

- Let's leave the group in place
- Remove the permission and restriction
- Remove the condition
- File: 09_DeleteRule.xml
- File: 10_DeleteCondition.xml



Summary

- We identified a granule as being 'bad'
- We create a restriction that applied to the 'bad' granule
- We created a group (and added ourself to it)
- We gave the group explicit permission to view the 'bad' granule



Example #9: Granting Permission to All Granules

- **The Set-Up:**
 - A provider has decided that all granules should be visible to their internal test team
 - The provider does not want to have to create a permission every time they restrict access to a granule



The Plan

- **Create a Permission against the group from before**
 - The permission should use the 'ALL' keyword
- **Issue a query to confirm that data is visible**
- **Attempt to hide the data by creating a restriction with the 'ALL' keyword**
- **Re-Issue the query to confirm that data is still visible**
- **Clean-Up**



Create a Permission

- **Create a Boolean Condition**
 - Set the value to 'true'
- **Create a Permission**
 - Reference Group# (from Example 8)
 - Use the 'ALL' keyword
- **File: 01_CreateBooleanCondition.xml**
- **File: 02_CreateGroup.xml**
- **File: 02_CreatePermission.xml**



Issue a Query

- **Look for Granules**
- **You will receive hits**
- **File: 03_Query.xml**



Create a Restriction

- **Use the Boolean Condition created for the Permission**
 - Condition re-use
- **Use the 'ALL' keyword in the Restriction**
- **File: 04_CreateRestriction.xml**



Re-Issue the Query

- Look for Granules
- You will receive hits
 - This is because you are a member of the group the permission has granted access to
 - If another user issued the query (who was not a member of the group), they would not receive any hits
- File: 05_Query.xml



Now Switch Users

- Logout from DemoUser(N)
- Login to DemoUser(N+1)
- Re-Issue the Query
- You will get no results
 - This is because DemoUser(N+1) is not in the group the permission references
- Don't forget to switch back to your user account
- File: 06_Query.xml



Clean-Up

- Delete your permission
- Delete your restriction
- Delete your condition
- Delete your group
- File: 07_DeleteRule.xml
- File: 08_DeleteCondition.xml
- File: 09_DeleteGroup.xml



Summary

- The provider does not want to have to create a Permission each time a Restriction is created
- A universal Permission was created to grant access to a specific group
- Members of the group will always have permission to view the metadata holdings of the provider.



Permission Wrap-Up

- Typically use Boolean Conditions
- Allow members of a group special access to a metadata holding that would otherwise be restricted
- Can apply to granules or collections
- Like restrictions, a boolean condition based permission is required to grant access to collection-level metadata



Overall Wrap-Up

- Conditions provide part of an evaluation mechanism (they are just phraselets)
 - Temporal
 - Rolling Temporal
 - Boolean
- Restrictions contain a Comparator, a Condition, and a piece of metadata.
 - They are rule-based, and evaluated at run-time
 - They apply to the entire ECHO community
- Permissions contain a Comparator, a Condition, a Group, and a piece of metadata.
 - Management of the group is separate from management of the metadata

