FUEL YOUR KNOWLEDGE.

NETWORK  LEARN  DISCOVER
Data Migration Done Right for Microsoft Dynamics 365/CRM

Daniel Cai
Principal Developer, KingswaySoft
Microsoft Business Solutions MVP
Meet the Presenter

Daniel Cai
9 Years of development experience with Microsoft Dynamics 365/CRM, with primary interests in
• CRM SDK Programming
• Large-scale ETL development
• Performance Tuning
• Anything that makes integration easier and more approachable

About KingswaySoft
A leading provider of data integration solutions for Microsoft Dynamics 365 software and beyond, with two primary focuses
• Connectivity
• Productivity
Disclaimer

Demo Code
Demo code is built on top of Microsoft Dynamics 365 SDK
• I assume some basic SDK knowledge and C# coding skills
• Credit goes to the community and the SDK team

Sample ETL Process
Sample ETL Process is built on top of SSIS Integration Toolkit for exhibition purpose
• The intention is to demonstrate the technical approach rather than advocating a particular product
• More details of SSIS Integration Toolkit can be found at the following URL
• The approach demonstrated should be transferrable to any other third-party or home-grown solutions in a similar fashion, as applicable
The Challenges
What are the challenges?

**Complex data migration requirements**
- Constantly evolving enterprise application landscape
- Disconnected data structure between the source and target systems

**Stakes in the project management**
An often overlooked piece of work in CRM projects
- Limited resources (capable resources)
- Tight schedule (often a last-minute rush)
- Lack of strategies and supporting resources

Many moving targets to aim at
- Constantly evolving CRM application metadata
- Frequent changes to integration/migration requirements

**Technical complexities**
- Many intricate details when working with Microsoft Dynamics 365/CRM API
- Web Service Interfaces are generally slow, getting to the right performance level can be a significant challenge
Planning
Strategize, Identify, and Plan
Migrate vs. In-Place Upgrades

In-Place Upgrade

Benefits
- Usually the preferred upgrade methodology if upgrading from a previous version
- All historical trace is brought over, including the original createdby, createdon, modifiedby, modifiedon, etc.
- All relationships are carried over, without having to work through the relationship tree in the case of using a data migration approach
- No data migration required

Drawbacks
- The number of hops can be extraordinary (i.e. an upgrade from v3 to Dynamics 365):
- The downtime can be significant due to the number of hops to go through
  - There are some db upgrades during the process which can take a very long time due to the significant schema changes
- Currently only possible for on-premise
  - Direct DB Import support might be possible in the future for Dynamics 365 Online.
- It is not guaranteed that every step will run successfully and smoothly (yes in most cases)

Migration

Usage scenarios and benefits
- Support a completely remodeling/refactoring of the application without having to be tied up to the original designs that may be flawed
- Organization restructuring, merge/acquisition or spinning off
- Can avoid the lengthy in-place upgrade process which may not be always a guaranteed success
- Maximum flexibility in designing the new application model

Drawbacks
- It is a challenge to keep all the historical trace – createdby, createdon, modifiedby, modifiedon
- The data load has to respect the relationship dependency and migrate in a proper order
- A full-scale migration without a proper tool can be extraordinarily difficult.
Writing to CRM database directly is simply not an option

Writing to CRM database can be very tempting, but it is very dangerous
- It can break data integrity.
- It can cause undesired application behaviors - such behaviors may not surface right away.
- Breach your CRM data security model.
- Potentially expose your system to SQL injections.
- Void your product warranty with Microsoft.

Don’t ever try to write to CRM database directly at any time!
- It’s the law!
- Make sure that this is made clear to your new CRM developers.
What are the options?

Build your own
Custom migration/integration development using CRM SDK

Drawbacks
• Resource intensive
• Many intricate details to deal with
• Creating a solution that is scalable can be a challenge
• It is not guaranteed to be a cost effective solution (most likely not)

CRM Import Data Wizard
• Designed for self-served data import
• Comes with many limitations in the area of lookup reference, error handling, troubleshooting, scheduling, etc.

ETL Tool (Commercial or Free)
Options
• Informatica
• Scribe
• Microsoft SQL Server Integration Services (SSIS) along with a third-party connector
• Many other alternative ETL tools

Problems or drawbacks
• Learning curve
• Licensing cost (if any)
How to plan for a successful migration?

- **Identify** your data sources
- **Identify** key stakeholders
- Define the scope
- Start with documented business **requirements**
- Look for opportunities to fix legacy design issues

- Create a **team** with competent technical resources including support from business stakeholders
- Choose the migration software and **technology**
- Build a repeatable and modular **process**
- Establish a baseline **infrastructure** to support the load

---

**Plan**

- **Develop** the mapping from source to the target system
- **Document** the ETL process
- **Fine-tune** the performance
- **Validate**, validate, and validate

**Prepare**

- Pre-migrate and test before the final production run
- Make sure key stakeholders are involved
- Ensure key milestones are clearly communicated to the team including the downtime planned
- Follow up with post-migration validation focusing on user experience

**Migrate**
How to evaluate the solution to use?

Scalability
- How does the solution scale? Horizontally, Vertically?
- Does the solution support multi-threading? What about bulk data load API (ExecuteMultiple)? A combination of both?
- Can the solution fully leverage system resources available?

Flexibility and Easy of Use
- How easy to get started with the solution? What kind of learning curve does it require for a Business Analyst? A developer?
- Any support of dynamic runtime values?
- How easy is it to create a formula or expression?

Robustness and Feature-richness
- How does the solution handle error situations? Any centralized job execution log?
- Any strategies to work around intermittent errors?
- Does the solution handle all the special cases, statecode, statuscode, multi-target lookup, support of using either integer value or label values for OptionSet fields?
- Does the solution support working with N:N relationship entities, principalobjectaccess? And support of close entities (opportunityclose, quoteclose, orderclose, incidentresolution) at the time when the primary entity is closed?
- Support of record fidelity (createdby, modifiedby fields)?
- How easy is it to work with lookup fields? Any chance to pass in a text value and the solution figures out the ID to be used for the lookup fields?

Cost of Ownership
- One time fee or evenly distributed subscription model
- Subsequent maintenance for the perpetual license model

Support
- How easy to find an answer for a particular error of the solution?
- How responsive is the team to respond to technical issues? Any presence of the team in social media channels (Dynamics Communities, MSDN Forum, Twitter, LinkedIn, Facebook)
- Any additional ticket based cost?
- What about data governance and privacy?
Data Integrity
Working with multi-target lookup fields

CRM lookup field value is made up of two parts
- The lookup entity’s name
- The lookup record’s primary key (GUID)

Missing one can cause the lookup to fail!
- SDK Code

```csharp
account.OwnerId = new EntityReference
{
    Id = ownerId,
    LogicalName = "systemuser",
};
```

- SSIS Mapping for ownerid field

<table>
<thead>
<tr>
<th>Input Column</th>
<th>Destination CRM Field</th>
<th>Text Lookup</th>
<th>Data Type</th>
<th>Create</th>
<th>Update</th>
<th>Unmap</th>
</tr>
</thead>
<tbody>
<tr>
<td>owneridname</td>
<td>ownerid</td>
<td>Yes</td>
<td>uniqueidentifier</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>owneridtype</td>
<td>owneridtype</td>
<td></td>
<td>nvarchar (64)</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Sorting out the relationship dependency

The problems

• CRM application is all about relationship.
• Many CRM entities have a circular reference.
• Writing to CRM without a proper sequence will make it inefficient and can cause data integrity issue.

Giving an example

• account entity has a parentaccountid field that points to account entity itself
• account entity has a primarycontactid field that points to contact entity
• contact entity has a parentcustomerid field that can point to either account, or contact entity

Solution: Use multiple passes in an efficient manner

• Migrate account entity without mapping the primarycontactid and parentaccountid fields
• Migrate contact entity
  • For those contact records that has an empty parentcustomerid, or a parentcustomerid that points to account, migrate with the parentcustomerid mapped
  • For those contact records that has a parentcustomerid that points to contact entity itself, migrate without having the parentcustomerid mapped, followed by an update task to update parentcustomerid field
• Update parentaccountid and primarycontactid for account if either of the fields is not empty.
Working with CRM lookup fields can be painful

The problems
• CRM stores all foreign keys with GUID values
• You don’t always have the GUID value for the lookup records
• Figuring out the GUID values based on an alternate business key values can be painful

Solution: Have an in-place lookup strategy
• The solution takes an input of the lookup business key values (usually text)
• The component should perform a lookup of the value and find out the corresponding record’s primary key (GUID) to be used for the lookup field.

How does it look like in SSIS
Avoid self-generated IDs

The problems
- CRM allows to use any primary key values (GUID) that you like
- When using self-generated IDs, they are fully randomized
- When the randomized GUIDs are saved, they can make the data storage very fragmented which can affect your CRM application query performance down the road

Solution: Have an in-place lookup strategy
- Leave the ID generation to the platform.
- The only exception would be data migration – in which case you can carry over the IDs from your legacy CRM system.

Further read:
The Dangers of Guid.NewGuid();
- http://blogs.msdn.com/b/crminthefield/archive/2015/01/19/the-dangers-of-gui...
Audit field carry over

The problems

CRM does not have a way to write to those audit fields directly

• createdby
• createdon
• modifiedby
• modifiedon

The solution

• To write to createdby, modifiedby fields, you will have to use service proxy impersonation.

```csharp
var serviceProxy = new OrganizationServiceProxy();
serviceProxy.CallerId = createByUserId;
serviceProxy.Execute(createRequest);
```

NOTE: Keep in mind, createdby can only be set at the time of creation, similarly modifiedby can only be set through UpdateRequest.

• To write to createdon, you can simply write to overriddencreatedon field.

```csharp
entity["overriddencreatedon"] = (DateTime)createOnValue;
```

• Writing to modifiedon field directly is not supported.

However it is possible to achieve it using a CRM plugin, you will have to user a holding field to receive values from the service call, and then in the plugin code, you pass the value to modifiedon field. This needs to happen in the Pre-Operation. This works programmatically or through an ETL tool if you have the plugin programmed and properly registered.
Close CRM entity records with a reason

The problem:

- A few CRM entities have an associated close entity which can be used to record the reason why primary entity records were disable/closed.

<table>
<thead>
<tr>
<th>Primary entity</th>
<th>Close entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>incident</td>
<td>incidentresolution</td>
</tr>
<tr>
<td>opportunity</td>
<td>opportunityclose</td>
</tr>
<tr>
<td>quote</td>
<td>quoteclose</td>
</tr>
<tr>
<td>salesorder</td>
<td>orderclose</td>
</tr>
</tbody>
</table>

The Solution:

```csharp
var opportunityCloseEntity = new Entity("opportunityclose")
{
    ["opportunityid"] = new EntityReference("opportunity", opportunityId),
    ["subject"] = (string)subject,
    ["actualrevenue"] = new Money((decimal)actualRevenue),
};

var winOpportunityRequest = new WinOpportunityRequest
{
    OpportunityClose = opportunityCloseEntity,
    Status = new OptionSetValue(newStatus),
};

serviceProxy.Execute(winOpportunityRequest);
```
Data Quality
Perfect timing for data quality assessment

The problems
- Poor data quality affects user adoption
- "Garbage in, garbage out" stands well in most implementations

Recommendations
Data migration is the perfect timing for a deep review of the current system’s data quality.
- Fix them before bringing the data to the new system
- Sometimes you have to use some special techniques such as data normalization, in some cases, maybe de-normalization if necessary.

It is also the perfect time for creating a set of rules or guidance to ensure future data quality going forward.
- User entered data
- Data brought in using a data integration software (ETL process)
Data Duplication

The problems:
• Duplicates happen and they can be a pain to deal with
• Duplicate data can significantly reduce team productivity
• Depending on the number of duplicates in the system, eliminating the duplicates can be a significant effort without a productive tool

The solutions:
• In-App Duplicate Detection
  • In CRM view, launch the Detect Duplicates function.
  • Define the characteristics of the duplicate detection job and create the job.
  • Wait for the job to complete.
  • Open the job details and work with the identified duplicates one at a time (the top set)
• The ETL approach (SSIS)
Error Handling
All errors happen for a reason

Business logic errors:
• Usually caused by some server side business logics.
• Can be fixed by ensuring input data to satisfy the business logics (such as required fields – the input should not be empty).

Data errors:
• Usually caused by wrong data type or incorrect data length or range.
• Can be fixed by ensuring the right data type with correct data length and range.

Intermittent errors:
• Happens intermittently.
• Usually caused by temporary network glitches or temporary server outage.
• A retry of the same service call can be attempted to help recover from the error without having to stop the job.

Authentication and permission errors
• Make sure the connection user can connect through web UI
• Make sure the user has right permissions in the system

Service timeout errors:
• When timeout error happens, there is a good chance that you have a design issue with your application (if the timeout setting is not too short)
• For reading, you would want to optimize your FetchXML query.
• For writing, you would want to use a smaller batch size. You would also want to take cascading rules into consideration.
• Increase Timeout setting with the service proxy.

Unhandled platform errors:
• Such as
  • Generic SQL error
  • Unexpected error
  • Internal Server Error
• The only way to troubleshoot unhandled platform errors is to turn on CRM trace log

```
orgServiceProxy.Timeout = timeout;
```
Performance Tuning
Start with something but look at the big picture

Performance tuning is a complex topic
- There are many things that can affect your data migration/integration performance
- Writing to CRM is slow due to the nature of web service operations

How to get it right?
- It is important to be strategic but also tactical
- Start with something as the baseline
- One step at a time to see the performance impact
- Always have a big picture in mind, as there are so many things that can affect your integration/migration performance
  - Don’t get stuck with just one thing.
- When there is a performance issue, there is a reason
Minimize the number of service calls

When creating

- When statecode/statuscode are involved, if the provided statecode is the default one, you can save a service call
- The ownerid can be posted with the same service call of the CreateRequest
  - If the current user is the owner, you can drop the field, it will perform faster

When updating

- It is important to leverage the enhanced Update operation so that the statecode/statuscode, ownerid fields are all submitted in one service call (Exception: opportunity, quote, salesorder, etc.)

```csharp
using (var service = new OrganizationService(crmConnection))
{
    Entity account = new Entity("account");
    account["accountid"] = new Guid("xxxxx-xxxx");
    account["name"] = "Adventure Works Inc.";
    account["creditlimit"] = new Money(100000);
    account["statecode"] = new OptionSetValue(1); //inactive
    account["statuscode"] = new OptionSetValue(2); //inactive
    account["ownerid"] = new EntityReference { LogicalName = "team", Id = new Guid("xxxxx-xxxx") };

    var request = new UpdateRequest() { Target = account };  //savemore
    var response = (UpdateResponse)service.Execute(request);
}
```
Use db indexes when necessary

Create db indexes when
- The migration job involves filtering query during the process
  - This applies to the situation that you need to perform Update or Upsert based on a natural key

How?
- For CRM on-prem, db indexes can be created using SSMS directly
- Alternatively, there are a couple other options available if it needs to be done within CRM application
  - Create an alternate key (requires CRM 2015 Update 1 or above) – this requires uniqueness
  - Add the column to the entity’s Quick Find view’s Find Columns
  - Use the Data Performance optimization wizard available by going to Settings > Administration > Data Performance.

What else should I know about db indexes?
- Having too many DB indexes can slow down your database write operation, don’t overuse it
- For those in-app options, there is usually a delay before the index is actually being created
- When creating indexes through SSMS, make sure the index follows certain naming convention and have it documented
- Avoid firing such events that might create db indexes during business hours – as it can bog down your system
Avoid submitting unchanged fields

Why?
- Smaller SOAP message, which in turn results in less networking time
- Can avoid firing unnecessary CRM plugins/workflows or auditing that are registered for the fields

How to achieve this?
- Have a way to compare changes, only post the fields that have changes
- If using SSIS, then just simply turn on "Ignore Unchanged Fields" option in the CRM destination component
Mind your network conditions

CRM Diagnostics Tool

Ideally the latency should be as low as possible
- Make sure the integration server (or workstation) is close to the CRM Server
- A poor infrastructure can result in high network latency, even the servers are close to each other

Bandwidth should be as high as possible
Performance Impact from Plugins/Workflows

The impact
• CRM plugins/workflows will have some performance impact depending on how they were designed.

How to minimize the impact?
• Disable Plugins/workflows for initial data migration if possible
• If using SSIS, enable "Ignore Unchanged Fields" option in CRM destination component
• Avoid any loop process within CRM plugins/workflows if possible at all
• When registering a plugin step, make sure to use Attribute Filtering option.
• Design plugins/workflows to run asynchronously as much as possible, and have a capable asynchronous processing server.

Register New Step

General Configuration Information

- Message: Update
- Primary Entity: account
- Secondary Entity: 

Filtering Attributes: All Attributes
Leverage CRM Bulk Data API

CRM Bulk Data Load API

- Helps improve data migration/integration performance
  - Particularly when the networking latency is high
  - Instead of sending one request at a time, it sends multiple requests within the same service call
  - When working with attachment entities (activity_mime_attachment, annotation), you want to be a little conservative (1 to be safe avoid potential server busy errors)

```csharp
var requestPackage = new ExecuteMultipleRequest
{
    Settings = new ExecuteMultipleSettings
    {
        ContinueOnError = false,
        ReturnResponses = true
    },
    Requests = new OrganizationRequestCollection(new[] { createRequest1, createRequest2 })
};
var executeMultipleResults = (ExecuteMultipleResponse)_serviceProxy.Execute(requestPackage);
```
Load in Parallel (Multi-threaded writing)

With a proper multi-threaded writing support, you can achieve a much greater performance with better flexibility.

Single destination, single thread

Use BDD (Balanced Data Distributor) or Conditional Split component to achieve multi-threaded writing.

With a proper multi-threaded writing support, you can achieve a much greater performance with better flexibility.
Choose the right combination of Batch Size and Multithreaded Writing

On-Premise

• The number of threads can be as high as your CRM/database server can sustain the load
  • Start with 20 and watch the server resource utilization, then decide whether to increase or decrease the number

• The Batch Size can be 100-200 for typical CRM entities
  • Use a smaller number if working with an attachment entity

CRM Online

ExecuteMultiple requests are throttled. At most 2 concurrent ExecuteMultiple requests allowed per organization.

• Use a batch size of 100, and set the number of concurrent threads to 2.
• Use a batch size of 10 (or lower) and then use a relatively high number of concurrent threads (start from 20 and then decide to increase or decrease depending on how the performance goes
• Raise a request with Microsoft Dynamics 365 support team to have the throttling lifted, then you can use a higher number of Batch Size along with the number of threads you have achieved a good result.
Multi-threaded vs. Single-threaded writing

Single Thread vs Multithreading

Writing to CRM 100,000 records, 100 batch size, using a Create action with a total of 5 fields

Single Thread
100,000 records in 4761 seconds
21 records/second

Multithreading (20x threads)
100,000 records in 425 seconds
235 records/second

11 times faster when using multithreaded writing

The performance testing was done in a laptop with everything installed in the same box, the processor was heavily throttled, it should perform a lot better with a real production system
It does not have to be one big BANG

There are many risks with the Big-Bang approach

- Limited time window
- Lack of thorough validations

What are the options?

Create a migration process that can bring in changes incrementally

Pilot run

Run legacy system in parallel and bring in more changes as needed

Final delta run

Design the package to use **Upsert** operation with "**Ignore Unchanged Fields**" option enabled

Do a pilot run to get the majority data in to the target system – which probably takes the most time

- Run the legacy system and new system in parallel and validate the data in new system
- As the validation processes, do one (or more) delta run to get incremental changes in the new system

Once the validation is done, do a final delta run to get all the delta changes into the final production system
Resources

- **White Papers**
  - Performance Benchmark for Microsoft Dynamics CRM Online 2016 Update 1
  - Microsoft Dynamics CRM 2016 Service Pack 1 Performance Benchmark on Azure Infrastructure as a Service (IaaS)
  - Microsoft Dynamics CRM 2015 White Papers & Technical Documentation
  - Optimizing and maintaining the performance of a Microsoft Dynamics CRM 2011 server infrastructure

- **Sample Migration Package**
  - Migration Starter Pack (based on SSIS Integration Toolkit for Microsoft Dynamics 365)

- **Blog Posts**
  - [http://www.kingswaysoft.com/blog/2016/09/16/Migrating-CRM-Activity-Entities-Properly](http://www.kingswaysoft.com/blog/2016/09/16/Migrating-CRM-Activity-Entities-Properly)
  - [http://www.kingswaysoft.com/blog/2017/03/31/Migrate-CRM-close-or-resolution-entity-records-in-your-ETL-process](http://www.kingswaysoft.com/blog/2017/03/31/Migrate-CRM-close-or-resolution-entity-records-in-your-ETL-process)

- **Best Practices**
  - Best practices for developing with Microsoft Dynamics 365
Thank You

Daniel Cai, Principal Developer
KingswaySoft
daniel.cai@kingswaysoft.com
@danielcai

Complete Session Survey
What did you think of this session? We value your feedback in our efforts to continuously improve, please complete the session survey.

#CollaborateCanada