UPDATE SETS

Get your development together.

Get your [stuff] together. Get it all together and put it in a back pack. All your [stuff]. So it's together.
—Morty

UPDATE SETS are used to track, maintain, and deploy development changes – whether it's configuration, code, layouts, or anything else that's tracked in Update Sets. Once your changes are ready for deployment to another instance, Update Sets are also how you move your changes between instances. You can export them to XML (once the State is set to Complete) and import that XML into the target instance, or you can **retrieve** them from the target instance before previewing and deploying them. Whichever method you use, it's important to *sanitize* your Update Sets, keep them clean, and keep them **functional**. That's "functional" as in "functional programming".

In this chapter, we're going to learn about **batching**, what is and **isn't** tracked in Update Sets, common pitfalls when dealing with scoped records, and how to correctly handle **preview** errors.

♦ ♦ ♦

UPDATE SET BATCHING

ServiceNow released **Update Set batching** in the Jakarta release of the Now platform, allowing you to **group** Update Sets together in a logical hierarchical grouping. This enables you to preview and commit them in bulk, and in order.

Update Set batching takes a lot of the hassle and annoyance out of managing large groups of Update Sets for a major release, and allows you to extract a specific set of functionality from a release without impacting the rest. For example, consider a scenario where you need to do **A,** which enables you to do **B**, which enables you to then do **C**. If there’s a problem with **B**, whoever does the deployment will have to be aware that **C** relies upon **B**, or the deployment of **C** will fail spectacularly!

"Dependencies" like this are a major issue, but also consider which of the records in your update sets might **reference** other records. For example, if you have a table which extends another table, you cannot push the **child** table before pushing the **parent** table, because the child references the parent table in order to inherit its fields and attributes.

Hierarchical Update Set batching makes this process easier, but is by no means a catch-all solution. You still have to **build**, **maintain**, and **control** the Update Set hierarchy and think carefully about where each Update Set belongs in the chain. Typically, the best-practice is to have the chain build chronologically, in order to avoid having a newer version of a record in a parent Update Set, and an older version of the record, in a child Update Set (which would result in an error).

If a child update set contains a record that also exists in one of the update sets that is “up-stream” from it, the version of that record in the child update set **must be newer** than the version in the parent set.

Pro-tip: *Deploying an update set will cause a system cache flush, which can impact performance. It is often a good idea to deploy during lunch, or before or after business hours.*

MASTER UPDATE SETS

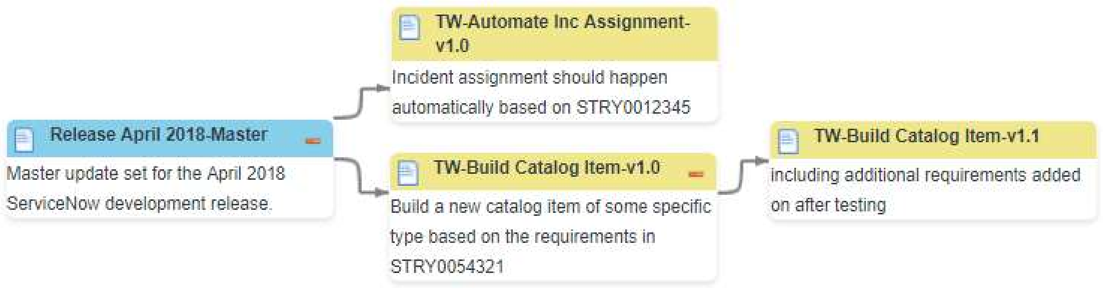
It’s not a bad idea to begin a release or large project by creating a “Master” Update Set in your dev environment. This is the Update Set under which all related "child" Update Sets will live. It’s also good to have a naming convention that makes sense, but that’s something your team will have to figure out internally to determine what works best. I'll use the name **Release April 2018-Master** as my example "master" Update Set.

No updates will actually go into the master Update Set. Instead, this will just be used to "contain" the Update Sets that contain any development that should be moved along as part of the "April 2018" release. Whenever a developer creates a new Update Set for some development, they'll set the **Parent** field on their Update Set to this master set.

As an example, imagine I've created two Update Sets for some development that I've been assigned stories for: **TW-Automate Inc Assignment-v1.0** and **TW-Build Catalog Item-v1.0**, and give them both helpful descriptions. I set these Update Sets' **Parent** field to **Release April 2018-Master**, since they're both scheduled for the April 2018 release, but if they needed to be pushed back at some point, I could give them a different parent. The master Update Set would not have a parent.

After identifying something additional that needs to be added, I create **TW-Build Catalog Item-v1.1**. This "1.1" Update Set will be a child of **TW-Build Catalog Item-v1.0** *rather than* **Release April 2018-Master**, since it is dependent directly on v1.0. This way, if the story to which they both relate needs to be moved back to the next release, I can just reassign v1.0, and v1.1 will come along with it. By making 1.1 the child of 1.0 rather than the child of the master set, I also ensure that the order of deployment is maintained, which is important because 1.1 will surely reference records contained within 1.0.

Let's have a look at what these relationships look like visually. You can click the **Visualize Update Set Batch** from any Update Set form, to see a visual representation of the relationships between your Update Sets and their hierarchy.

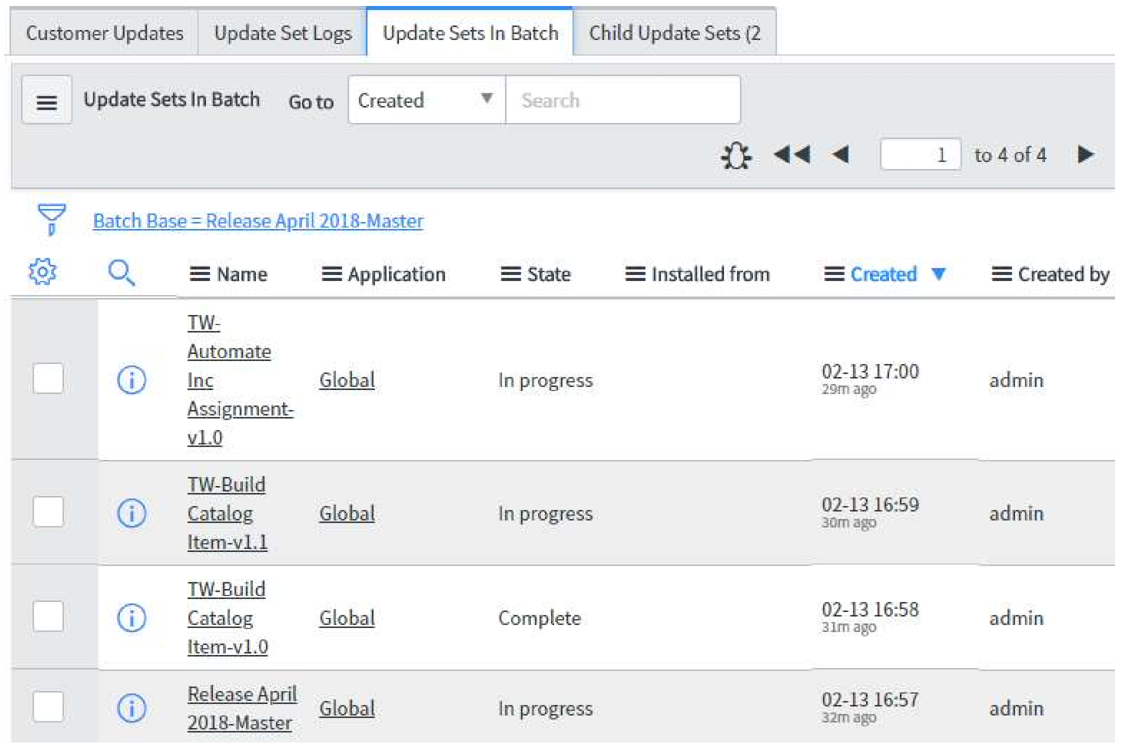

Release April 2018-Master
Master update set for the April 2018 ServiceNow development release.
TW-Automate Inc Assignment-v1.0
Incident assignment should happen automatically based on STRY0012345
TW-Build Catalog Item-v1.0
Build a new catalog item of some specific type based on the requirements in STRY0054321
TW-Build Catalog Item-v1.1
including additional requirements added on after testing

Note: *Right-clicking an Update Set in this view, and clicking* ***Prune****, will only prune it from that* ***view****. This does not actually prune the Update Set from the batch hierarchy. If you refresh the batch visualizer, the pruned Update Set will reappear.*

As you can see, from the master upset set, we split into two branches; but you can have as many child Update Sets as you like. If we then right-click on the master Update

Set, and click **Edit**, we can see the **Child Update Sets** related list. This is a list of all of the Update Sets which are **immediate** children of the master set. It does not include the *children of the children* of the master set.

To see **all** of the Update Sets in the entire batch, go to the maser Update Set, and open the **Update Sets In Batch** related list.



Batch Base = Release April 2018-Master

Name	Application	State	Installed from	Created	Created by
TW-Automate Inc Assignment-v1.0	Global	In progress		02-13 17:00 29m ago	admin
TW-Build Catalog Item-v1.1	Global	In progress		02-13 16:59 30m ago	admin
TW-Build Catalog Item-v1.0	Global	Complete		02-13 16:58 31m ago	admin
Release April 2018-Master	Global	In progress		02-13 16:57 32m ago	admin

Note: *Closing the master Update Set will close* ***all child Update Sets in the batch****. However, closing any child Update Set – even if it has children of its own – will not close any others in the batch, including its children.*

WHAT IS AND ISN'T TRACKED

In ServiceNow, it's important to understand the difference between **data** and **configuration/customization**. Data is stuff that fills up tables.

Only what ServiceNow considers **configuration/customization** is captured in Update Sets, but it can sometimes be a little bit difficult to pin down what constitutes configuration without simply knowing in advance. For this reason, if you're not 100% certain that a particular record type is indeed captured in Update Sets, it's a good idea to open your Update Set in another tab, sort your updates by the *sys_updated* column, and make sure that your changes were captured.

Records which are not "captured" in Update Sets can still be forced into them. For example, if you want to force a specific record in the **Location** [cmn_location] table (which isn't tracked) into your Update Set, you can do so in a Background Script using the GlideUpdateManager2 API. In some cases, such as with a large number of records, it makes more sense just to export them all to XML and manually import them into the target instance after deploying your Update Set.

Pro-tip: *To make it a little easier to include data records (since nearly every developer needs to do this quite often), you can download a tool I wrote, which puts a "Include in Update Set" button in the Related Links of every record and list context menu for every table that isn't tracked in Update Sets if you're an admin. You can find this free tool at http://updatetracker.snc.guru/*!

While **not exhaustive**, here is a list of some of the records that are/aren't included in update sets. You might be surprised by some of the examples.

The following **are** tracked in Update Sets:

- Roles
- Form sections and layouts
- Reports
- Form and list views

The following are **not** tracked in Update Sets:

- Scheduled jobs & scheduled script executions
- Schedules
- Configuration Items
- Users
- Groups
- Group memberships
- Role assignment (to users *or* groups)
- Homepages/Dashboards

You'll also find that there are certain records that are dependent on other records for their functionality. For example, if you create a role in the **Role** [sys_user_role] table, that will be tracked. If you then create a **group** (which isn't tracked), add the **role** to that group, and then force that group into your update set, you might be surprised to find that the roles you gave it **didn't come along with it**. This is because the associations between roles and groups is stored in the **sys_group_has_role** table which is also not tracked in update sets. Similarly, the association between **users** and roles is in the **sys_user_has_role** table, which is also not tracked in update sets.

There is a similar situation that happens with **dashboards**. Dashboards contain several linked *pieces* that are each not captured in update sets. Each new tab in the **pa_tabs** table is linked to two tables (**sys_portal_page** and **sys_grid_canvas**). To get dashboards into your update set, you have to grab each of those types of records.

TRACKING SCOPED RECORDS

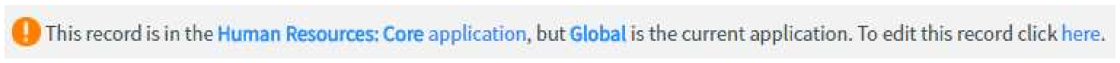
To be honest, working with Update Sets inside of scoped applications in ServiceNow can be a real bother. In this section, we'll go over a few specific rules to be aware of, to hopefully save you and your team some headaches.

For starters, it's important to understand that a single Update Set should never contain records from multiple application scopes. The platform tries to prevent this, but for some reason, it happens anyway. If at all possible, you want to avoid it at all costs. Update Sets have a specific scope, and you cannot deploy updates in one scope using an Update Set in another scope.

There are plenty of scenarios where you may need to work within multiple scopes. For example, the **HR** application consists of multiple separate application scopes including **Human Resources: Core** and **Human Resources: Service Portal**. One development effort may require that you work with records in both HR scopes, as well as records in the Global scope. This typically requires manually creating **three separate Update Sets**: one for each scope. The Update Set you have selected persists when you switch back to a specific scope, but not between scopes.

For example, if you have **Update Set 1** selected in the **Human Resources: Core** scope, then you switch to the **Global** scope and select **Update Set 2**, and finally switch *back* to the **Human Resources: Core** scope, you'll find that you still have **Update Set 1** selected. This will persist until you change your Update Set *while in that scope*.

There are some additional headaches to contend with, too. For example: typically, when you visit a record that's in a different scope than your session is in, you cannot edit the record and you get a warning like this:


This record is in the Human Resources: Core application, but Global is the current application. To edit this record click here.

You can click the link at the end of that message to switch (for the duration of one update) to the scope in which the record resides, but you won't be able to see what

Update Set is selected in that scope unless you switch your whole session to the relevant scope.

Another reason to be very careful when dealing with scoped updates, is that some **tables** are in a certain scope, but the **records** in them are in a different scope. Again, ServiceNow is working on this, but when this happens, there is no good way to tell that you're effectively poisoning your Update Set with updates in multiple scopes, making it so it won't be able to be deployed.

For an example of this, consider the following steps:

1. Select the **Human Resources: Core** application
2. Create an Update Set in the **Human Resources: Core** application. Set it as your current Update Set.
3. Navigate to the **sn_hr_core_service** table.
4. Open any record in that table, and make any update to it, so the record is captured in your Update Set.

You'll notice that the record is tracked in your Update Set. However, in some instances, you'll also notice that even though the **sn_hr_core_service** table (like your Update Set) is in the **Human Resources: Core** scope, the update itself is in the **Global** scope. This Update Set cannot be deployed until all updates it contains are in the same scope that it's in.

Unfortunately, the only way around this right now when dealing with multiple scopes, is to double-check that any new updates you make are tracked in your Update Set, and in the correct scope.

Note: *ServiceNow has been making an effort to patch this issue wherever they find it by updating the plugin, but they do not usually offer retroactive patches if you've already deployed a given scoped application or plugin. You might have some luck if you notice this issue in your instance, by opening a HI ticket (http://hi.servicenow.com/). The key is just to keep an eye on it to make sure your Update Set and update scopes match.*