WHAT IS YOUR APP ‘BEST AT’?
DESIGNING FOR WINDOWS 8 WITH
THE METRO DESIGN LANGUAGE

Francis Beaudet, Senior Interaction Designer
COMING WITH IT IS MICROSOFT’S soon-to-be-released new design language called ‘Metro.’ Metro’s been around for a while, but in Windows 8, it makes the jump from touch-screen interface to the desktop.

Metro changes the landscape for developers, giving them a true, touch-optimized design language for Windows applications.

With Metro comes strict behaviour and visual guidelines, some of which are based on User-Centered Design principles. The visual guidelines are pretty clear and teams could concentrate on those to create apps. They would end up with apps that look just like Metro apps, but if your team doesn’t also follow the behaviour guidelines, your app won’t have the Metro DNA. And Microsoft will enforce their behaviour guidelines through their Windows Store review process.

So, if you want to create an app using the Metro design language, your team will be following these five steps:

- Create a ‘Best at’ statement;
- Choose the right scenarios;
- Pick a navigation pattern;
- Lay out the content; and
- Create app bars.
Step One: Create a ‘Best at’ statement

No team sets out to create mediocre software. You want to dominate your category. The place to begin is to decide what the app will do—but not only do, but do best.

This isn’t just a marketing exercise. It might seem like one, but your team will use the Best At statement constantly throughout the development process. It’s a tool that keeps everything laser focused on the goal. Whenever your team makes a design decision, the sanity check is “Does this support the Best At statement?”

How to create the best ‘Best At’ statement

All your team has to do is fill in the blank:

The app is the best app in its category at ______.

Easy, right?

Not at all. Chances are that you, or your team members, already have an idea of the app and some great features it should have. To get the best Best At statement, the team has to lose all of that and start from zero.

A good Best At statement includes phrases that limit scope by identifying the users, their task, and the timeframe.

For example, a great Best At statement might look like this:

The app is the best app in its category at helping users and their friends find a restaurant to eat at tonight.

It’s not easy to get rid of preconceived ideas. Your team might need to throw out plans they’ve already made (even if they’re just in the pre-planning stages).

Thing is that the Best At statement needs to be a corkscrew, not a Swiss Army knife. The Best At statement does one thing only. Like a corkscrew, it will open wine bottles. It won’t open wine bottles, cut ropes, and file your nails.

To keep the app on track, the Best At statement must be focused, specific, and truly differentiated. Avoid commas and semicolons. Beware of the words ‘and’ and ‘or,’ though they can work, but probably not to connect verbs. They need to limit the app, not extend it.

Our Best At example statement provides boundaries for the features that will be brainstormed in step five. It identifies a target user group, pinpoints the goal of the users, and limits the scope of the app with a timeframe. Our statement does contain the word ‘and’ (“helping users and their friends”), but in this case, the ‘and’ actually limits the app because it shows the focus on the social aspect of the task. For this app, the ‘and’ does not create an additional scenario—it weeds out scenarios where the user decides on restaurants on their own.

Step Two: Choose the right scenarios

Even the best Best At statement will fail if the team adds scenarios that don’t support it.

Picking the right scenarios isn’t hard in theory; it’s only hard in practice.

The team chooses the right scenarios for the app by brainstorming. Same way they would do any brainstorming session, with no boundaries to the suggestions. (If your team doesn’t have an established brainstorming process, you might want to check out the brainstorming section of our whitepaper The Software Manager’s Guide To Practical Innovation.) After that, the team prunes scenarios down to only the ones that get closer to the Best At statement.

The team can start by taking all the features and putting them aside for later. This isn’t the time to come up with features. If anything on the brainstorming list looks like a feature, they should save it for step five.

At this point, the team wants usage scenarios, not features. Stories of what users will do with the app, not tools that the app provides. These stories must support the Best At statement and will show how users will use the app, not what tools will be available for them.

In the end, the team should finish with 2-3 interacting related narratives that will be the core of the app.

Step Three: Pick a navigation pattern

Once the team has the scenarios, the next thing they need to do is choose the right navigation pattern for the job.

Here’s where the Metro differences come in.

The whole Metro philosophy is ‘Content over chrome.’ By ‘chrome,’ Microsoft doesn’t mean Google’s web browser. They mean all the UI elements that take up the borders of your screen. Metro is based in the concept of as little UI as possible. Most of the UI of a Metro app is hidden off-screen, brought in with a mouse or finger gesture.

The design of Metro itself took a lot of inspiration from the kinds of maps in malls and airports that show where other places are, but their biggest feature is a blazing badge that says “You Are Here.”

It’s a natural navigation pattern, based on the idea that in order to know where you’re going, you have to understand where you are. From there, apps offer users other destinations based on the scenarios identified in the brainstorming session.

Two navigation patterns

Metro offers two navigation patterns: Hierarchical and Flat.
Hierarchal patterns concentrate on the concept of a hub, sections, and then detail pages.

The hub is the front-facing view of the app, and is where all the personality of the app should be. It’s the equivalent of the Start menu or launch point.

On the second level, there are the sections, which act as exhaustive lists of the content contained within them. One of the key differences between a hub and a section is that with the hub, you pick and choose the content to show. It may not be chronological, organized into groups, or have any other logic, but once the user gets into a section, everything has to be findable.

The detail is where the user reads the final content when they reach their destination.

Hierarchical patterns are good for apps with large collections of content to explore, and multiple sections/categories. In Windows itself, the Start menu uses a hierarchical pattern. Other apps that use a hierarchical pattern would include stores and news services.

The flat pattern is more of a free-form pattern that concentrates on the content. One example of a flat organization pattern is tabs in a browser. The flat pattern is the “go to” pattern when the app deals with the user switching context: documents, chat conversations, TV channels. Ideally, in a flat pattern, the app has no chrome at all. All the UI is hidden off screen. In order to access the UI, the user would use a gesture.

Step Four: Lay out the content

After the team has identified the best pattern, the next step is laying out content in a logical way. In this step, the team fits scenarios into the hub, section, and detail levels (if they’re using a hierarchical pattern).

The biggest help here is the starting template that ships with Visual Studio. It comes with pre-built hub, section, and detail levels.

Metro provides many different ways you can lay out your content, and we won’t be able to go into the details of that in this introduction article. There will be as many ways to lay out content as there are apps. Teams will get to use their creativity and knowledge of their user to create layouts that support the scenarios they identified in step two. For example, one possible strategy is to take each scenario and create a different group in the hub to support that scenario.

The most important thing is that teams remember their sanity check: Does this layout put us closer to the Best At statement? That is the key to every design decision in a Metro app.

Step Five: Create app bars

If the team has followed the steps to here, they won’t have identified any features yet. They will just have the scenarios they brainstormed in step two. Now it’s finally time to come up with some features.

An example of an app bar. From left to right: View-switching commands, contextual commands, and global commands.

We suggest brainstorming again. And when it comes time to prune out features, there are extra criteria for a Metro app. The team should only keep features that:

• Make one of the scenarios great;
• Most users will use; and
• Are specific.

When pruning features, the team should:
• Remove features that don’t align with scenarios;
• Remove features that make use of “charms,” like system menus. In a Metro-style app, searches, logins, sharing content—these system-level things are all done by swiping in off-screen features. Apps don’t need to worry about stuff like Start, Search, Share, Devices, and Settings;
• Remove features that are laid out as part of the content. If the content does the equivalent of the feature on its own, then the app doesn’t need that feature. For example, organizing content by category will happen naturally when the team sets up the section level; and
• Windows 8 provides a context menu to cut, copy, and paste, so the app doesn’t need those features.

Once the team has a list of features, they sort them according to the scenario they belong to and group them into related sets.

Arranging features
The four keys to great feature set placement are:
• Think about the context;
• Keep command sets in consistent positions throughout the app;
• Separate distinct sets using the edges or separators; and
• Follow the conventions for common feature sets.

Those conventions are:
• The “New/Add/Start/Create/Compose” command uses the + icon and is on the far right;
• Commands that show on selection appear on the far left;
• The “Accept” command uses the ✔ icon; The “Reject” command uses the ☑ icon. Accept appears to the left of Reject; and
• View switching commands appear on the left.

On the app bar, keep command sets in consistent positions all the way through the app. Essentially, global commands belong on the right; contextual commands stay on the left. Commands that relate to the selection belong even further to the left.

A radical shift for Windows design
The Metro design language, with its strict look and behaviour, forces application teams to concentrate on specific scenarios over lists of features. It will help teams to create narrow, focused apps without the feature bloat that Windows programs have been known for.

This proposed step-by-step guide to creating your Metro application uses some basic User-Centered Design principles to help you focus on what is important to your user.

Making developers follow some guidelines of UCD in order to have their Windows 8 apps available in the Windows Store could just force a radical shift in the usability of Windows software.

We’ll find out as more companies begin designing for Windows 8.