Table of Contents

CRM Developer Guidelines

Created by Microsoft

Amended by Ahmed Elsawalhy  
(Link Development)

V1.4

[1 Document change control 1](#_Toc20764673)

[2 Objective 2](#_Toc20764674)

[3 Assessment Process & Tools 2](#_Toc20764675)

[3.1 Common Quality Attributes 2](#_Toc20764676)

[4 Application Lifecycle Recommendations 5](#_Toc20764677)

[4.1 Requirement and Work-Item Tracking 5](#_Toc20764678)

[4.2 Prototyping 5](#_Toc20764679)

[4.3 Version Control 5](#_Toc20764680)

[4.4 Integrated Development 7](#_Toc20764681)

[4.4.1 Document hardware and software requirements for developer workstations 9](#_Toc20764682)

[4.5 Definition of Done 9](#_Toc20764683)

[4.6 Deployment 10](#_Toc20764684)

[4.7 Testing Best Practices 10](#_Toc20764685)

[4.7.1 Acceptance Criteria & Testing Strategy 10](#_Toc20764686)

[4.7.2 Testing Areas 11](#_Toc20764687)

[4.8 Defect Management 12](#_Toc20764688)

[4.9 Operations Best Practices 13](#_Toc20764689)

[4.9.1 Transition to Support 14](#_Toc20764690)

[4.9.2 Monitoring and Tuning 14](#_Toc20764691)

[4.9.3 Supporting Production and Future Development 17](#_Toc20764692)

[4.9.4 Update Rollups 19](#_Toc20764693)

[5 Implementation Recommendations 20](#_Toc20764694)

[5.1 Configure First, Customize (Code) Second 20](#_Toc20764695)

[5.2 Configuration of Microsoft Dynamics CRM 21](#_Toc20764696)

[5.2.1 Solutions & Publishers 21](#_Toc20764697)

[5.2.2 Entities & Relationships 22](#_Toc20764698)

[5.2.3 User Interface 27](#_Toc20764699)

[5.2.4 Processes 29](#_Toc20764700)

[5.2.5 Security Model 32](#_Toc20764701)

[5.3 Extending Microsoft Dynamics CRM 35](#_Toc20764702)

[5.3.1 Client Side Scripting – JavaScript 35](#_Toc20764703)

[5.3.2 Plug-ins 43](#_Toc20764704)

[5.3.3 Custom Workflow Activities 59](#_Toc20764705)

[5.3.4 Ribbon 59](#_Toc20764706)

[5.3.5 Site Map 60](#_Toc20764707)

[5.4 CRM Online Considerations 61](#_Toc20764708)

[6 LINK Development Naming Amendment 63](#_Toc20764709)

[6.1 Processes 63](#_Toc20764710)

[6.2 Plugins 63](#_Toc20764711)

[6.3 Fields 63](#_Toc20764712)

[6.4 Relations 63](#_Toc20764713)

[6.5 Web Resources 64](#_Toc20764714)

[7 LINK Development Developer Amendment 64](#_Toc20764715)

[7.1 Coding 64](#_Toc20764716)

[7.1.1 General 64](#_Toc20764717)

[7.1.2 Plugins 65](#_Toc20764718)

[7.1.3 JS 66](#_Toc20764719)

[7.2 Customisations 67](#_Toc20764720)

[7.3 Security 69](#_Toc20764721)

[7.4 Tools 70](#_Toc20764722)

[8 LINK Development UX Amendment 70](#_Toc20764723)

[8.1 Design Target 70](#_Toc20764724)

[8.1.1 Design for the specialist role 70](#_Toc20764725)

[8.1.2 Design for the business user 70](#_Toc20764726)

[8.2 Dashboards 72](#_Toc20764727)

[8.2.1 Purpose 72](#_Toc20764728)

[8.2.2 Guiding questions 72](#_Toc20764729)

[8.3 Navigation Bar 73](#_Toc20764730)

[8.3.1 Purpose 73](#_Toc20764731)

[8.3.2 Design 73](#_Toc20764732)

[8.4 Forms 74](#_Toc20764733)

[8.4.1 Design and purpose 74](#_Toc20764734)

[8.4.2 Layout 74](#_Toc20764735)

[8.4.3 Fields 74](#_Toc20764736)

[8.4.4 Quick create forms 75](#_Toc20764737)

[8.5 Views 76](#_Toc20764738)

[8.5.1 Quick view forms 76](#_Toc20764739)

[8.6 Grids 77](#_Toc20764740)

[8.6.1 Sub-grids 77](#_Toc20764741)

[8.6.2 Associated grids 77](#_Toc20764742)

[8.6.3 Editable grids 77](#_Toc20764743)

[8.7 Business Process Flow 78](#_Toc20764744)

[8.8 Performance 79](#_Toc20764745)

[8.8.1 Avoid scripting 79](#_Toc20764746)

[8.8.2 OnLoad event 79](#_Toc20764747)

[8.8.3 Server call-back functions 79](#_Toc20764748)

[8.8.4 Plugins 79](#_Toc20764749)

[8.9 Theming 80](#_Toc20764750)

[9 Appendix: Code Reviews 81](#_Toc20764751)

[9.1 Code Review Entry Criteria 81](#_Toc20764752)

[9.2 Code Review Exit Criteria 81](#_Toc20764753)

[9.3 Review Suggestions 81](#_Toc20764754)

[9.3.1 Code structure 81](#_Toc20764755)

[9.3.2 Change Control 82](#_Toc20764756)

[9.3.3 Performance/Optimization 82](#_Toc20764757)

[9.3.4 Exception Handling 82](#_Toc20764758)

[9.3.5 Variables 82](#_Toc20764759)

[9.3.6 Execution Control 82](#_Toc20764760)

[9.4 Suggested Questions 83](#_Toc20764761)

[9.4.1 Maintenance 83](#_Toc20764762)

[9.4.2 Coupling 83](#_Toc20764763)

[9.4.3 Cohesion 83](#_Toc20764764)

[9.4.4 Reuse 84](#_Toc20764765)

[9.4.5 Performance 84](#_Toc20764766)

[9.4.6 Correctness/Reliability 84](#_Toc20764767)

[10 References 86](#_Toc20764768)

# Document change control

|  |  |  |  |
| --- | --- | --- | --- |
| Revision Number | Date of Issue | Author(s) | Brief Description of Change |
| 1.2 | 18 / Jul / 2017 | Ahmed el-Sawalhy | Added an amendment for UX guidelines by LINK Development |
| 1.3 | 10 / Aug / 2017 | Ahmed el-Sawalhy | Added an amendment for developer guidelines by LINK Development |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Objective

CONTOSO is adopting Microsoft Dynamics CRM as an enterprise platform to host their different business solutions. The successful adoption of Microsoft Dynamics CRM is driven by the processes used by the organization to design, build, operate and govern the platform.

This document provides a **guideline** to proven recommended practices and methodologies for Microsoft Dynamics CRM to prepare CONTOSO for its successful adoption as an enterprise solution platform, and to drive platform consistency and governance. The concepts discussed in this document must be carefully reviewed by the project teams at CONTOSO and tailored to fit their unique needs.

The Solution Design Document presents the solution’s design that typically applies to production. This current document focus on the implementation of the solution’s design into the development environment that will be deployed to other environments including tests, pre-production and production.

# Assessment Process & Tools

This section describes an iterative technique that we used to think about and sketch out your architecture. It helped bring together the key decisions and issues discussed during the assessment; including quality attributes, architecture styles, application types, technologies, and deployment decisions. The technique includes a series of five main steps, each of which breaks down into individual considerations.

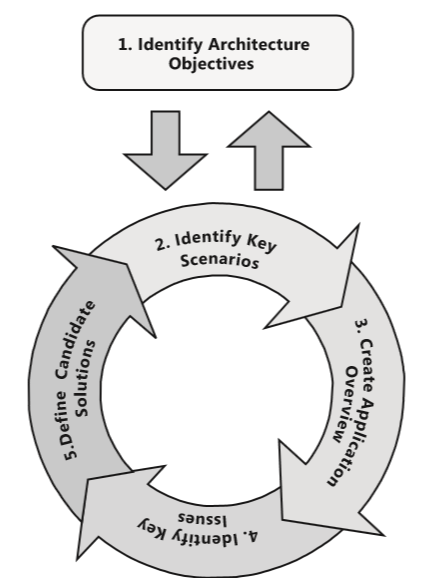


Figure - Iterative evaluation technique

## Common Quality Attributes

Quality attributes are the overall factors that affect run-time behavior, system design, and user experience. They represent areas of concern that have the potential for application wide impact across layers and tiers. Some of these attributes are related to the overall system design, while others are specific to run time, design time, or user centric issues. The extent to which the application possesses a desired combination of quality attributes such as usability, performance, reliability, and security indicates the success of the design and the overall quality of the software application. When designing applications to meet any of the quality attributes requirements, it is necessary to consider the potential impact on other requirements.

We analyzed the tradeoffs between multiple quality attributes. The importance or priority of each quality attribute differs from system to system; for example, interoperability will often be less important in a single use packaged retail application than in a line of business (LOB) system.

This table lists and describes the quality attributes that we considered when evaluating the design of the solution. The table will help you gain an understanding of how quality attributes map to system and application quality factors. You can use the sections containing key guidelines for each of the quality attributes to understand how that attribute has an impact on your design.

|  |  |  |
| --- | --- | --- |
| Category | Quality Attribute | Description |
| Design Qualities | Conceptual Integrity | Conceptual integrity defines the consistency and coherence of the overall design. This includes the way that components or modules are designed, as well as factors such as coding style and variable naming. |
| Maintainability | Maintainability is the ability of the system to undergo changes with a degree of ease. These changes could impact components, services, features, and interfaces when adding or changing the functionality, fixing errors, and meeting new business requirements. |
| Reusability | Reusability defines the capability for components and subsystems to be suitable for use in other applications and in other scenarios. Reusability minimizes the duplication of components and also the implementation time. |
| Run‑time Quality | Availability | Availability defines the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period. Availability will be affected by system errors, infrastructure problems, malicious attacks, and system load |
| Interoperability | Interoperability is the ability of a system or different systems to operate successfully by communicating and exchanging information with other external systems written and run by external parties. An interoperable system makes it easier to exchange and reuse information internally as well as externally. |
| Manageability | Manageability defines how easy it is for system administra- tors to manage the application, usually through sufficient and useful instrumentation exposed for use in monitoring systems and for debugging and performance tuning. |
| Performance | Performance is an indication of the responsiveness of a system to execute any action within a given time interval. It can be measured in terms of latency or throughput. Latency is the time taken to respond to any event. Throughput is the number of events that take place within a given amount of time. |
| Reliability | Reliability is the ability of a system to remain operational over time. Reliability is measured as the probability that a system will not fail to perform its intended functions over a specified time interval. |
| Scalability | Scalability is ability of a system to either handle increases in load without impact on the performance of the system, or the ability to be readily enlarged. |
| Security | Security is the capability of a system to prevent malicious or accidental actions outside of the designed usage, and to prevent disclosure or loss of information. A secure system aims to protect assets and prevent unauthorized modification of information. |
| System Qualities | Supportability | Supportability is the ability of the system to provide information helpful for identifying and resolving issues when it fails to work correctly. |
| Testability | Testability is a measure of how easy it is to create test criteria for the system and its components, and to execute these tests in order to determine if the criteria are met. Good testability makes it more likely that faults in a system can be isolated in a timely and effective manner |

# Application Lifecycle Recommendations

In this section, key development best practices are highlighted. These recommendations are designed to ensure optimal development efficiency and minimize conflict between development team members when working with Microsoft Dynamics CRM.

## Requirement and Work-Item Tracking

Typical development processes require various types of information: requirements, tasks, bugs, tests, etc. Central management of this information is crucial for enterprise CRM development. The Visual Studio environment and TFS work-item tracking provides enterprise-level support for tracking, assigning and connecting development work-items.

Dynamics CRM deliveries using the Dynamics Sure Step methodology will typically produce a base Work Breakdown Structure (WBS) as the output from the design phase. TFS work-item schema and the Excel import-export feature can be used to import the requirement list and create the initial WBS even on larger CRM projects. In addition to the tasks identified for each work item, the following two additional tasks must be added:

* Unit Test: This task ensures that the developer spends adequate time unit testing the item they just completed.
* Peer Review: It is a good practice to assign a different developer to peer review the work-item completed by their colleague. This ensures that a comprehensive review of the build items has taken place to achieve optimal code quality.

## Prototyping

When designing a new CRM organization, it is important to prototype it with considerations of the product’s possibilities. For this, it is recommended to create the following organizations for each team:

* “Vanilla” containing out-of-the-box functionalities as a reference that will not be customized
* A prototype sandbox organization to try any customization that needs to be validated with users (ideas, proof of concepts, examples). This organization support the specification process and improve communication with stakeholders and users. It is also entirely independent of all official customizations done by the project team.

## Version Control

One of the most fundamental challenges in any enterprise development project is the control, tracking, and management of changes within source code files.

From a CRM solution perspective, the following items can be considered as source elements:

* CRM solution content (stored in XML format)
* CRM base data
* Web resource files
* .NET source files
* CRM database scripts

The detailed content of the above elements is illustrated in the following figure:

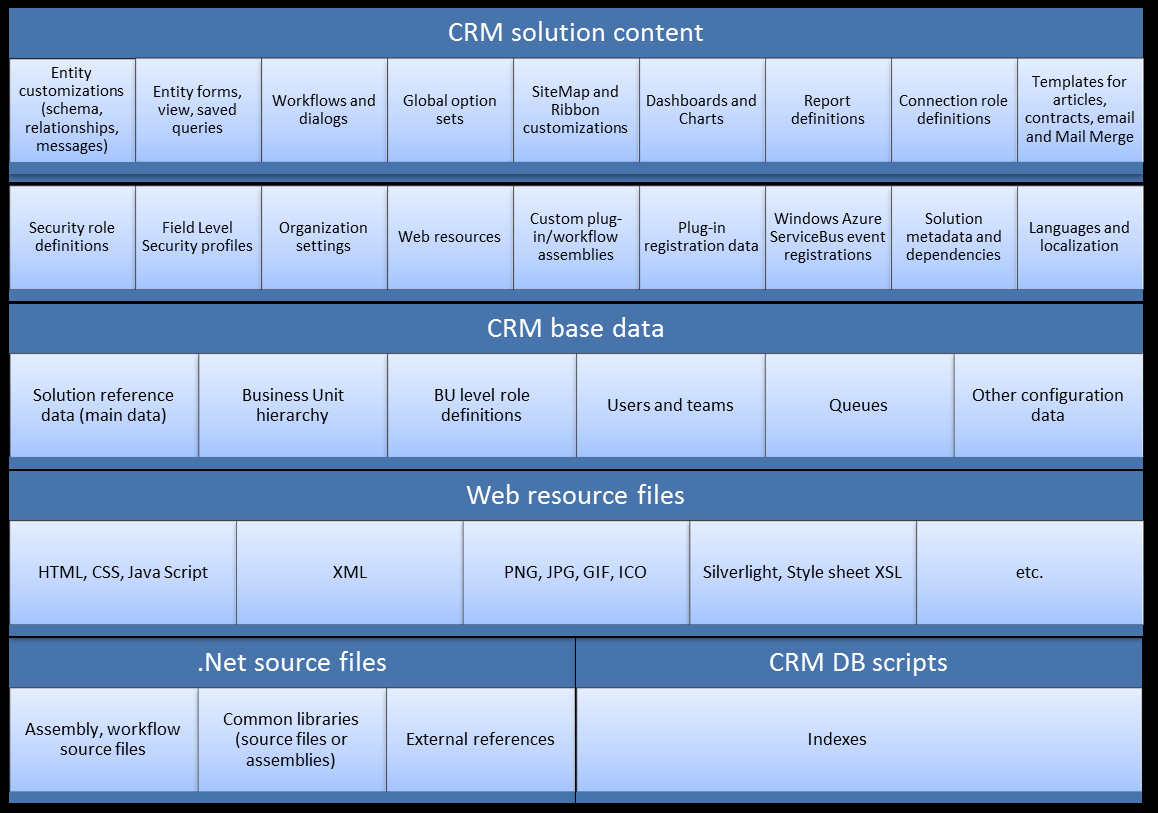


Figure - CRM Implementation Components

To be able to store and version control the individual components of CRM solutions, the XML files need to be split into basic elements and stored discretely under version control. This approach supports the individual tracking and change control of components within a CRM solution. The Solution Packager Tool (available [here](http://msdn.microsoft.com/en-us/library/jj602987.aspx)) is used to pack and unpack a CRM solution into its individual elements. The unpacking of solution XML files need to be executed for every version change to CRM configuration and tracked via version control. In a developer environment, the unpacked solution would need to be packed together with any binaries built in order to create a CRM solution to deploy within their development environment. The packing operation is also required during a team build to create the CRM solution package for deployment.

## Integrated Development

There are a number of topologies that can be used when setting up a development environment for Microsoft Dynamics CRM. The following setup is recommended for CONTOSO: ***To review depending on plans for Dev/Test on Azure and actual location of PROD***

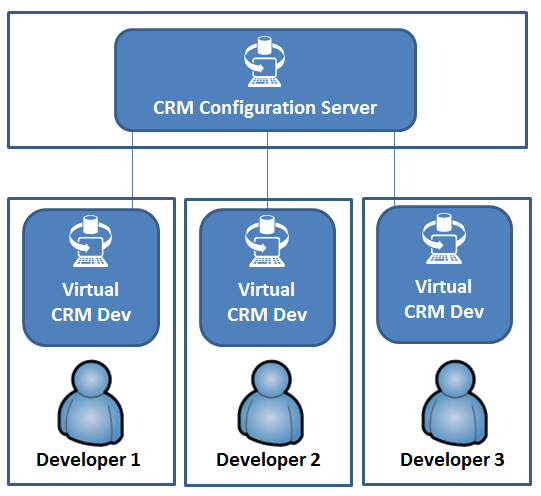


Figure - Multiple CRM instances, locally Hosted with central configuration organization

In this scenario, a centrally managed deployment is used to host the integrated configuration organization. This is where all activities to create entities and relationships (schema) take place. It is recommended that a designated member of the development team (or a Business Analyst) be the owner of the schema in the configuration organization. They are responsible for all configuration activities to setup the solution entities.

The developer works on their local environment to complete the CRM development task allocated. Before starting a work item the developer (or the build master) gets the latest version of customization tree from TFS and uses the pack tool to create the solution zip file. The developer (or the build master) deploys the latest solution package on the development environment.

After finishing a work item, the developer exports the solution zip file from the development environment. Executing the unpack tool on the zip file will generate the separate customization elements of the entire solution. The customization files need to be checked out and checked-in individually only when changed.

Some of the advantages that this approach provides:

* Complete developer isolation.
* Fast “onboarding” of a new developer; just copy the virtual CRM development machine.
* Ability to snapshot the complete virtual machine and restore to a known state.
* Central configuration organization is kept clean from trial development work.
* Developer is free to create CRM organizations etc. for complex trials in their own environment.

The daily development process is summarized in the following figure:



The above guidelines enable a development team to collaborate and work effectively together without creating conflict situations, misplacing or losing code. It is worth noting, however, that with team maturity, some of the steps recommended in this section might be combined or eliminated to achieve optimal efficiency in execution. The idea here is that there is no “one size fits all” process. Instead, these concepts should be adapted to the unique development culture that is in place at CONTOSO.

### Document hardware and software requirements for developer workstations

It is important to define a consistent developer workstation specification in terms of both hardware and software, including specific versions and service packs. This allows applications to be developed and built within a tried and trusted environment and increases developer efficiency.

As an example, a typical software list is:

* Admin rights + CRM Deployment rights
* Windows Server 2012 with IIS8
* SQL Server Dev Edition (change setting to reduce maximum memory limit)
* CRM Server
  + with latest Update Rollup and required organization(s)
  + with required language packs
  + (change registry setting for debugging plug-ins)
* Visual Studio 2012 with CRM Developer Toolkit extension
* CRM SDK folder
* Office (Outlook + Excel + Word)
* Target browser(s) (use the browser’s F12 or Fiddler for JavaScript debugging)
* PowerShell IDE
* SSRS (for report designers)

When needed, developers will also be able to import popular development solutions in CRM, such as OData Query Designer (in Dynamics XRM Tools), Visual Ribbon Editor, Ribbon Workbench, Fiddler, Diagnostics Tool.

## Definition of Done

A definition of done (also known as “Done-Done”) is important to get a shared understanding of expectations that software must live up to in order to be releasable into production (without obsessing too much for an exhaustive list of criteria). User stories and their list of related tasks per user story must be carefully planned by the team, and this done-done checklist can also help with pre-implementation activities for estimation and design. For example, a user story is completed when:

* The user story has been fully implemented (entity configurations in the master organization and code including comments).
* Final labels and messages are translated (in French and English) and configured.
* Static code analysis has been performed.
* Deployment scripts and related data (reference data and configurations) have been updated as needed.
* Unit tests with required code coverage have passed (automated, when applicable).
* Code review and test review have been performed.
* Code and changes have been checked in and associated with the user story.
* Code has been built and deployed to a test server.
* Test cases have passed.
* Documentation (list needed) has been updated.
* Product Owner has tested and signed off the requirement.

Above is an example of a done-done checklist for a user story and similar checklists could be prepared for a sprint (collection of features developed within a sprint) and a release (potentially shippable state). Of course, the checklists are not static and will evolve over time.

## Deployment

For all environments downstream of development, the smallest unit of deployment is a CRM solution package. No changes should be applied to any environment directly. All changes must be packaged in a CRM solution, checked into source control, and release notes detailing the changes must be prepared and checked into source control as well along with the CRM solution. The build manager then applies the deployment on the required environments. This discipline is key to track what changes are applied to which environments.

There may be other external components or applications that need to be deployed alongside the CRM solution for the CRM application to function correctly as a whole. It is recommended to automate the deployment process for all downstream environments and use the same approach throughout to test the process that will ultimately be used for production. A number of automated deployment approaches exist. The most typical for an enterprise production solution are Windows PowerShell scripted installations. These are the recommended approach to enterprise scale Dynamics CRM application deployments.

Windows PowerShell can be used in conjunction with .Net applications developed by the project team to orchestrate the full deployment of the Dynamics CRM application including, for example:

* Import of the Dynamics CRM solutions.
* Import of reference data.
* Registration of plug-ins.
* Deployment of bespoke, custom web services within IIS.

## Testing Best Practices

Identifying defects as early as possible is the least expensive way to ensure software quality. Best practices and tools can help the team minimize the cost of preventing and fixing defects by maintaining the quality of the project throughout its life cycle.

The following sections describe some of the testing best practices that must be considered when deploying any new business solution into the Microsoft Dynamics CRM platform.

### Acceptance Criteria & Testing Strategy

During the Analysis Phase, the QA lead must define the Acceptance Criteria and Testing Strategy that will be followed during the implementation of the new Microsoft Dynamics CRM business solution. The Acceptance Criteria defines the conditions upon which the delivered solution is accepted into production. This includes areas such as defect prioritization system, acceptable defect thresholds, and any other quality considerations that must be met to deliver a quality solution. The Testing Strategy defines how the business solution will be tested. This includes the iterations of testing required, functional areas to be tested in each iteration, as well as required environments for testing.

### Testing Areas

In a typical implementation of Microsoft Dynamics CRM, the following are the areas that must be thoroughly tested and verified before the business solution is deployed onto the platform.

You should always test any customizations you make with a user account that doesn’t have customization privileges. This way you can make sure that people without the System Administrator or System Customizer security roles will be able to use your customizations. To do this effectively, you need access to two user accounts: One account with the System Administrator security role and another that has the security roles that represent the people who will be using the customizations.

*Important:* Don’t attempt to remove your System Administrator security role if you have only one user account. The system will warn you if you try, but if you proceed you could find that you won’t be able to get it back. Most security roles don’t allow editing of a user’s security roles.

#### System/Functional Testing

Functional Testing focuses on validating the functional behavior of the business solution against the documented design. It is strongly recommended that the QA Lead be part of the design workshops to help define Test Scripts that are more aligned with the business expectations. These Test Scripts are based on the detailed design documents and must clearly identify the design element from the design document it tests (Entity Form, Client Script, Workflow, etc…), the expected behavior, the steps to execute as well as any required setup data/instructions needed to execute the script.

For unit tests of plug-ins, refer to section [Plug-ins Unit Tests](#_Plug-ins_Unit_Tests) in this document.

#### Data Migration Testing

When the implementation of a new business solution includes the migration of data from external sources to Microsoft Dynamics CRM, it is important that Data Migration Testing activities be part of the Testing work stream.

It is recommended that the Data Migration Test Lead be part of the design discussions to gain the valuable insight that will help in shaping the Data Migration test scripts. Data Migration activities and testing is typically executed in an environment other than that used for Functional Testing. This is to minimize the impact of running data migration scripts and tests on functional testing activities. Depending on the complexity of the business solution and the data migration requirements, this can simply be another organization in the Functional Testing environment, or an isolated environment completely separate from all other testing environments.

Also, in an iterative delivery approach, Data Migration testing activities are typically executed in later iterations once the schema of the business solution is well defined. This helps minimize the re-factor needed in the Data Migration scripts as a result of schema changes in the Microsoft Dynamics CRM solution.

#### Integrated Testing

Integrated Testing focuses on the end-to-end testing of the business solution in context with the other systems it interfaces with. Test scripts based on the interface design documents must be developed and executed to validate the integrated behavior of the business solution. As is the case with Data Migration testing, Integrated Testing typically takes place in the later iterations of the execution phase of the project. This ensures that most of the functional areas have been stabilized and verified to work in a stand-alone mode before the more complex task of testing them in an integrated end-to-end scenario.

For simple business solutions with no interface to external systems, this area of testing can be skipped.

#### UAT Testing

User Acceptance Testing (UAT) verifies that the business solution meets end user expectations and helps them do their daily jobs, and is designed to minimize any risks related to user adoption. Key end users must be involved in the design workshops and in the review of the UAT Test Scripts to ensure alignment in expectations. As is the case with the above areas of testing, UAT Test Scripts must clearly identify the designed behavior to aid the end user in their testing activities.

Another recommendation for successful UAT Testing is to conduct training sessions in the Analysis Phase of the project to familiarize the key stakeholders with the Microsoft Dynamics CRM platform. This is important to help shape the end user’s mindset and expectations, and educate them on what is possible within the tool.

#### Performance Testing

Performance Testing verifies the business solution conformance to the non-functional requirements defined for the platform. It is recommended that Performance Testing be executed in an environment with architecture and capacity that mirrors production. This verifies that the test results are as accurate as it can be to what the business solution’s performance will look like in production.

The amount and structure of data in the data source of an application will have a major impact on performance. Therefore it is crucial that your test data is as similar to the production database as possible.

The profile of the load used to drive the system during performance testing must be a realistic reflection of the load expected in production. Considerations include usage patterns, type of operations (read and write), numbers of users and think times between activities.

To facilitate the load testing of the performance of Microsoft Dynamics CRM deployments, it is recommended to use the Performance Toolkit for Microsoft Dynamics CRM. It contains the tools that can be used in customizing the CRM installation, populating the necessary semantic data for the deployment that is preferred and conducting the benchmarking tests against the CRM installation. More details about the Performance Toolkit can be found here.

By carefully planning the required dataset that the deployment needs to support and the workload requirements, the toolkit can be used to test if the scale and performance requirements of a particular deployment can be met. This methodology can be used to help with decisions on a particular deployment solution and to avoid costly downtimes at a later stage.

## Defect Management

Efficient processes to manage defect tracking and assignment is a key driver of testing efficiency and success. There are a number of different tools available that provide defect management capabilities, however, it is recommended to adopt Microsoft Team Foundation Server (TFS) to manage their defects. TFS provides a central repository of all work-items as well as defects that will provide the project team with a holistic view of the progress of the implementation. It also provides the delivery team a one-stop tool to manage their daily tasks (defects or build items). In addition, defects logged in TFS can be linked to work-items which helps in analyzing and tracking back to the build artifacts that caused the defect.

The figure below summarizes the steps recommended in the lifecycle of managing defects:



Figure – Defect Management Lifecycle

1. The CRM Tester identifies a potential defect during their testing activities. They log the defect in TFS with steps to re-produce it and screenshots if possible.
2. The QA Lead must triage these defects. The defects must not be assigned directly to developers by the CRM Tester. This ensures that the QA Lead had a chance to confirm that this is a defect that requires attention and sets the priority and due date on it.
3. Once the defect is confirmed by the QA Lead and have been prioritized, it gets assigned to the Development Lead/CRM Architect for analysis. The lead analyzes the defect, identifies the build artifact that needs to be re-visited, updates the defect with their comments and assigns it to the appropriate CRM Developer to be fixed. This is typically the same developer who built this item, but can be different if needed.
4. The CRM Developer receives the defect and it appears in the view of defects assigned to them. The defect in TFS contains all the information the developer needs to reproduce, isolate and fix it. Once the developer completes the fix for the defect, they attach any special release instructions, update the defect status and send it back to the development lead.
5. The development lead is responsible to coordinate the defect release with the QA lead to make sure minimal interruption to on-going testing activities.

## Operations Best Practices

Once the business solution is delivered and onboarded onto the Microsoft Dynamics CRM platform, it is transitioned to the support team. This section describes some of the best practices to transition, optimize and support the production business solution.

### Transition to Support

To successfully transition the business solution to the support team, it is recommended that the support leads be identified and engaged as early as the Analysis Phase of the delivery. This ensures that the support leads have visibility into the scope of the business solution they will be supporting. The support leads are required to deliver a transition and support strategy outlining:

* The transition approach and requirements (deliverables, knowledge transfer, operations guide, etc...)
* Roles and responsibilities for the dedicated support team
* Training requirements

In the Design Phase, the support leads are recommended to attend the design workshops to get detailed insight into the functional areas they will be supporting.

The business solution is formally transitioned to the identified support team during the Operations Phase of the project. This is established through:

* A series of knowledge transfer sessions and code walk-throughs conducted by the delivery team. In these sessions all configuration and customization elements are reviewed with the support team.
* All project documentation, including detailed design documents and the updated Operations Guide is transitioned and reviewed with the support team.

### Monitoring and Tuning

Continuous monitoring and tuning of the Microsoft Dynamics CRM platform is essential to optimizing the health of the platform. Once a business solution is deployed into productionmonitoring system logs is key to capture and address any potential issues. Microsoft Dynamics CRM lets you create trace files that monitor the actions that are performed by Microsoft Dynamics CRM. Trace files are helpful when you have to troubleshoot error messages or other issues in your environment. There are different levels of tracing that can be enabled (deployment vs. server); each determines the range of server roles monitored, the degree of control and how to enable tracing and the location of the trace files. It must be noted however that there are performance considerations that must be carefully evaluated and planned for when tracing is enabled for Microsoft Dynamics CRM; more details about enabling tracing in Microsoft Dynamics CRM can be found [here](http://msdn.microsoft.com/en-us/library/hh699694.aspx).

In addition to server trace files, Microsoft Dynamics CRM keeps a log of all jobs that are running in the Organization. These include logs of running workflows as well as system jobs running asynchronously in the background. These must be monitored regularly to ensure that raised errors and failed workflows are addressed. System Jobs logs can be queried by creating an Advanced Find against the System Jobs entity in Microsoft Dynamics CRM (see below). Columns can be added to the view to display details about when the job was started, its status, and any error messages it raised.

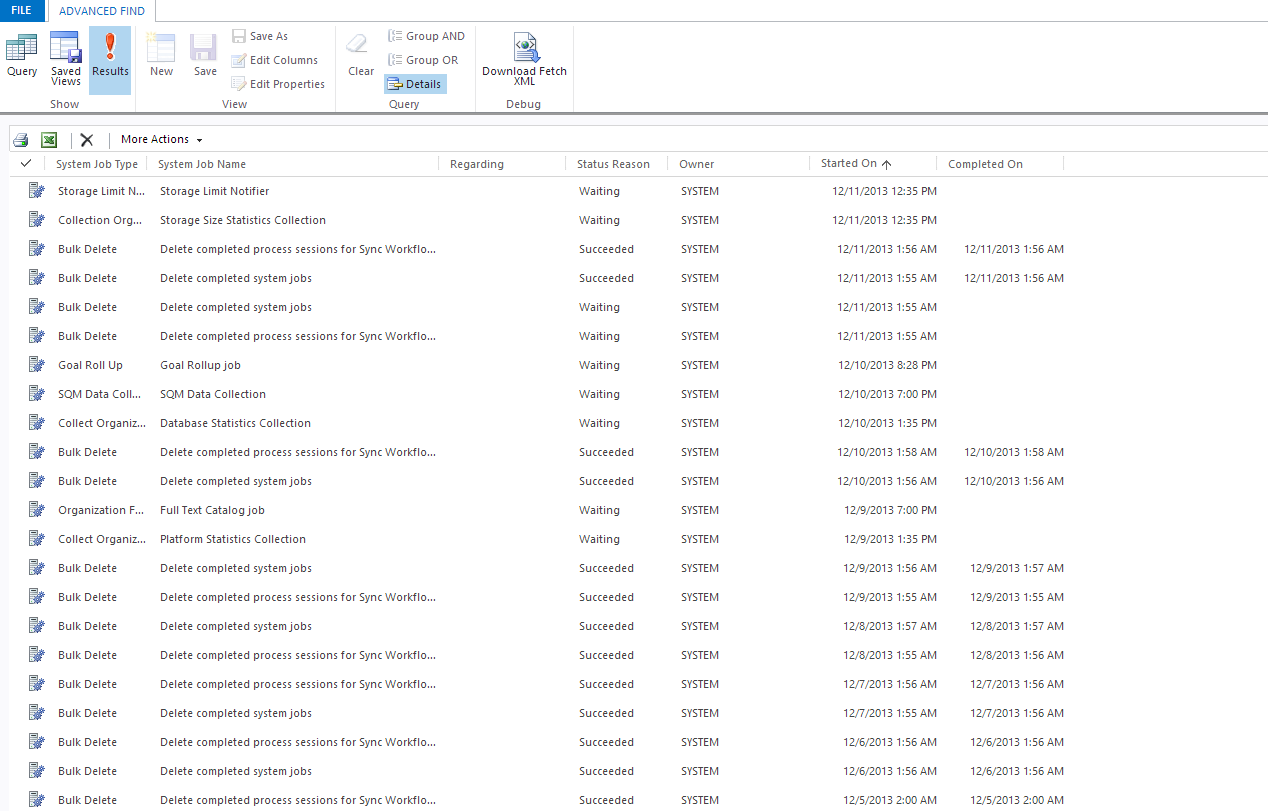


Figure 9 - System Jobs Advanced Find

These logs are stored in the Microsoft Dynamics CRM database and can grow very rapidly impacting overall performance and disk space. Scheduled clean up jobs must be created and registered to regularly archive completed System Jobs logs in a separate database and then delete the entries from the Microsoft Dynamics CRM database using the supported SDK methods.

You can also configure the workflows you add to your Microsoft Dynamics CRM business solution to automatically delete completed workflow jobs to save disk space. This can be configured in the Administration tab of the workflow editor as illustrated below

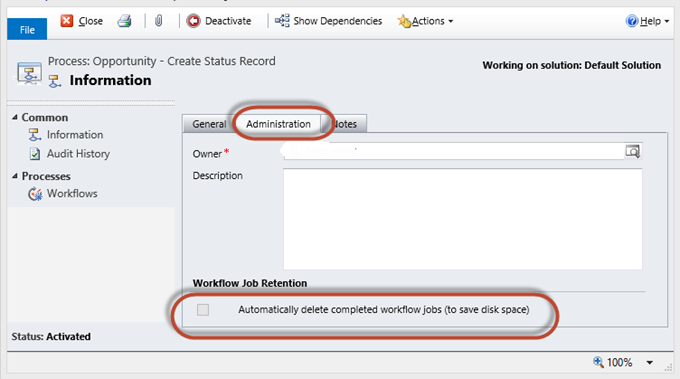


Figure - Workflow Job Retention

Whether to automatically delete completed workflow jobs must be a decision that is made on a case by case basis. In some cases, it is important to keep a log of completed workflow jobs, for tracking or compliance purposes. In other cases, once the workflow is stabilized in production, logs of its completed runs does not add great value and the option to automatically delete completed jobs can be turned on.

Another logging area that requires regular maintenance is Audit Logs. Microsoft Dynamics CRM provides the capability to audit changes to data and keep a trail of these changes in the database. Depending on how many fields are enabled for auditing, and the volume of activity in the business solution, these audit tables can grow very rapidly which might cause timeouts in retrieving the audit information and the overall performance of the business solution. As is the case with System Jobs logs, it is recommended to schedule jobs that will archive audit data older than a specified period of time in a separate database and then remove the audit partitions from the Microsoft Dynamics CRM Organization.

It is strongly recommended that the application support team include these monitoring and fine-tuning best practices in their support strategy to improve the performance of the Microsoft Dynamics CRM platform. Additional optimization techniques such as custom SQL indexes and IIS compression can be found [here](http://msdn.microsoft.com/en-us/library/jj126126.aspx).

### Supporting Production and Future Development

In any enterprise application, it is important that code base supporting production is kept separate from code base that is part of future development. Once a business solution is deployed into production, it is recommended that major development activities following the production release be performed in a separate environment. The figure below illustrates the general concept.



A separate track for major future development activities is needed. Each environment in the diagram above (DEV, QA, UAT) represents a black box representation of all the components that make the overall solution (Dynamics CRM, SharePoint, etc…). In some cases, these can simply be CRM tenants/organizations sharing the same infrastructure. When new releases are deployed to production (Future Development Track), the release has to be applied back to the production support track to ensure code consistency. Similarly, when a Hotfix is needed in production, the Production Support track is used and the Hotfix must be released to the Future Development Track after it’s released to production.



To provide for version control and source code management, it is recommended to use Microsoft’s Team Foundation Server (TFS) in addition to the Solution Packager utility for Dynamics CRM. More detailed information about application lifecycle management techniques for Dynamics CRM can be found in this whitepaper “ALM for Microsoft Dynamics CRM 2011: CRM Solution Lifecycle Management” [here](http://www.microsoft.com/en-us/download/confirmation.aspx?id=39044).

In some cases, separate tracks for future development and production support is not feasible. These cases should be handled with care. Governance processes must be put in place to ensure that no production support code and future development code is misplaced or deployed to the wrong environment. These situations should be handled on a case by case basis.

### Update Rollups

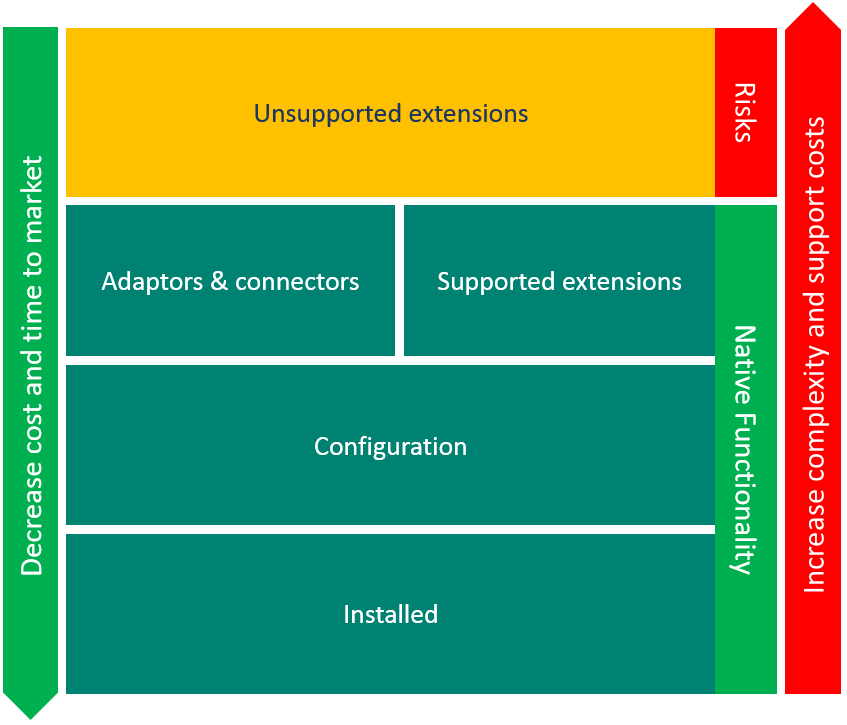
Investments in Microsoft Dynamics CRM are continuously made with yearly release cadence and Update Rollups delivered regularely. The support team in collaboration with business stakeholders must evaluate these updates for their fit before they are applied to the Microsoft Dynamics CRM platform.

It is recommended that these Update Rollups be applied to a sandbox environment first to perform initial testing of their impact on the platform. Since these updates are applied at the server level, they impact all the business solutions on the platform, and as a result, testing should cover all the business solutions. Once the Update Rollup is evaluated, and the impact of its rollout is assessed, if the decision is made to move forward with the update, it must go through the same stages as any other change:

* Development: The Update Rollup must first be applied to the development environment where unit testing is performed. Any re-factoring required as a result of the update must be identified and assigned to developers to work on.
* QA: Once the update is verified in the development environment and all re-factoring work-items have been completed, the update is applied to the QA environment. The QA leads for the different business solutions must identify high-impact functional areas that must undergo regression testing to ensure that the update and the re-factored artifacts did not introduce new defects.
* UAT: After regression testing is complete, the changes are applied to the UAT environment for key end users to validate that the business solutions still meet their acceptance criteria.
* Production: Once all testing is complete for all the business solutions, the Update Rollup is applied to production. It is recommended that the support team increase the level of monitoring right after the update, to rapidly identify and address any potential issues.

# Implementation Recommendations

## Configure First, Customize (Code) Second



Prioritize configurations of CRM before adding custom code. The law of instrument states “If all you have is a hammer, everything looks like a nail.” With all the different customization capabilities available in Microsoft Dynamics CRM it is easy to become very familiar with one of them and seek to use it to solve every problem. As you evaluate the business problems you want to solve, think about the end result you want to achieve and then work backwards to how you can get there. It is suggested to consider any change with the following options in this order:

1. Entities, Fields, Forms, Relationships and Business Process Flow
2. [Business Rules](https://technet.microsoft.com/en-us/library/dn531086(v=crm.6).aspx)
3. Processes and Workflows
4. Custom Coding
   * JavaScript (always using the supported XRM library)
   * Plugins
   * Web Resources (remember that Iframes and Web Resources will be not be displayed on phone and tablet apps)

In some cases, also consider the security models, including Business Units, Security Roles and Field security profiles.

***IMPORTANT:*** ensure that every developer understand the following pages regarding supported and unsupported customizations:

* [Customization concepts](https://technet.microsoft.com/en-us/library/dn531193(v=crm.6).aspx) (including section: Common unsupported customization practices)
* [Supported extensions for Microsoft Dynamics CRM](https://msdn.microsoft.com/en-us/library/gg328350(v=crm.8).aspx) (CRM 2016 SDK)

## Configuration of Microsoft Dynamics CRM

### Solutions & Publishers

Configuration and customization in the Microsoft Dynamics CRM platform are packaged into “solutions.” These solutions allow you to export configuration and customization artifacts from one environment, and import them into another environment.

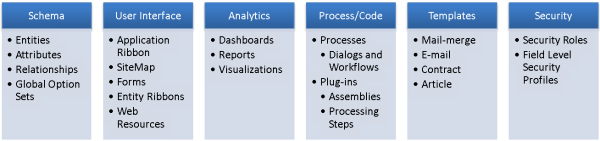


Figure - Dynamics CRM Solution Components

Solutions enable you to package:

* Entities – specifically, the following:
  + Global settings of the entity
  + Attributes of the entity
  + Forms
  + Views
  + Charts
  + Relationships with other entities
* Option Sets
* Extensions of the client application:
  + The site map
  + The global ribbon of the application
* Web resources:
  + HTML pages
  + JavaScript scripts
  + Images
  + Style files
  + Silverlight resources
* Processes:
  + Workflows
  + Dialogs
  + Actions
  + Business Process Flows
* Plug-in assemblies and SDK Message Processing Steps
* Dashboards
* Reports
* Connection roles
* Templates (Email, Mail Merge, Article, etc.)
* Security roles
* Field security profiles

It is a good practice to perform your configuration and customization in a dedicated solution for the business application you’re building. Avoid applying customizations directly in the undamaged system customization area of Dynamics CRM (Settings 🡪 Customization 🡪 Customize the System).

Also, when creating solutions, it is possible to create multiple solutions to store the various configuration and customization artifacts. This is particularly important when there is a large number of configuration and customization required in a particular line of business organization. Careful consideration must be taken when designing which configuration components are stored in which solutions to avoid complex inter-solution dependency that could introduce deployment complexities when importing the solutions to another environment. For example, if two solutions are created in the development environment with components in each solution that depend on other components in the other solution, this might cause import conflicts and/or sequencing challenges when the solutions are imported to another environment.

Another recommended practice when working with solutions is to create a dedicated Publisher for the line of business organization that will be used later on when importing a solution to another environment to identify the authoring publisher of the solution. Publishers in Dynamics CRM also allows to define certain settings that will apply to all the configurations inside the solution. An example is the prefix added to the name of any configuration item created in the solution. If a ‘Contoso\_’ prefix is defined in the solution’s publisher, any configuration added to the solution will be prefixed with ‘Contoso\_’, example ‘Contoso\_myCustomEntity’.

### Entities & Relationships

#### Entities

Customize a system entity, such as the opportunity entity, instead of replacing it with a new custom entity so that you can use the many built-in features in an existing entity. For example, the opportunity and case entities have lookup fields to associate customers. Customers may be accounts or contacts. You cannot create a custom entity that has the same type of lookup. You can change the display name of a system entity to make it more meaningful to your business.

***Apply naming conventions to custom Microsoft Dynamics CRM entities.***

The schema name of the entity will begin with the identified prefix that is defined by the publisher of the solution where the customization is created. In our case, the name will begin with “Contoso\_”— for example: Contoso\_country, Contoso\_ecifcrossreference. Note: schema names cannot be changed after the component is created.

***Define the ownership settings upon the creation of a Microsoft Dynamics CRM entity.***

Because the ownership settings cannot be modified after the entity has been created, it is important to evaluate the consequences of defining the ownership settings at this point.

The two levels of ownership of a Microsoft Dynamics CRM entity are the following:

* Ownership at an Organization level—the records of this entity are not owned by a known user, and access to the data is at an Organization level. With this level of ownership, a user can either see all records of this entity, or none. This level of ownership is typically used for globally accessible entities; reference data is an example.

Example: entity Contoso\_country

* Ownership at a team or user level—this is the default level for a business entity that has complete access-rights management.

***Define whether the entity is an Activity type in Microsoft Dynamics CRM upon creation.***

An entity can be defined as an Activity type, and treated by Microsoft Dynamics CRM like the other out of the box activity types (Phone Call, Task, Email, etc.). Because this setting cannot be modified after the entity is created, this must be evaluated carefully at the time of creation. To change this, the entity must be deleted entirely from Microsoft Dynamics CRM and re-created. This obviously has data loss and referential integrity implications and is typically a costly operation.

***Verify the compliance of the Primary Field attribute of the Microsoft Dynamics CRM entity.***

The Primary Field attribute of a CRM entity is the field that appears in the title of the form for that entity (example Contact’s Full Name title on the Contact form). It also appears in the Lookup field for the entity on any other form (example Contact’s Full Name appearing in the Contact Lookup on the form of a related entity). By default, the Primary Field attribute of a newly created entity will have Contoso\_name as its schema name. The Primary Field of an entity must be specified when the entity is created and cannot be changed.

***Activate the Connections option only if it is required.***

Because this option cannot be disabled, activate the Connections option only if it is required to link this entity to other entities.

***Activate the Activities option only if it is required.***

Because this option cannot be disabled, enable the Activities option only if it is required to associate activities to this entity.

***Activate the Notes option (includes attachments) only if it is required.***

Because this option cannot be disabled, it is important to enable the Notes option only if it is likely that you will have to relate notes and attachments to this entity.

***Activate the Queues option only if it is required.***

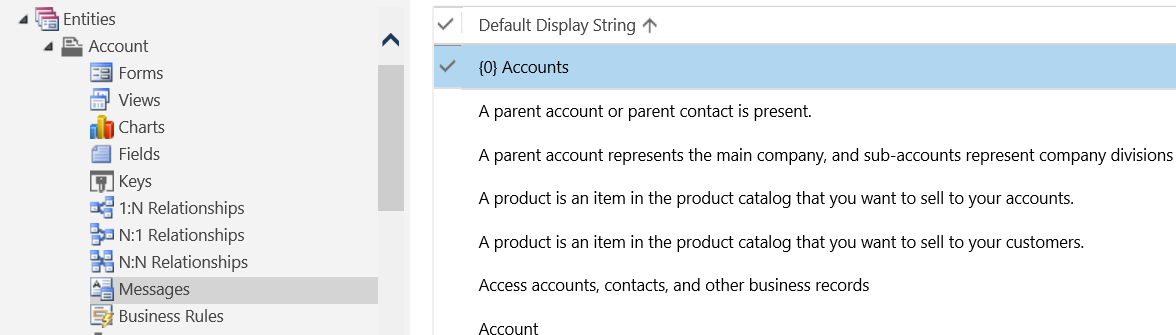
The Queues option allows you to place entities in a position of awaiting specific data processing from a user or a team (example Cases in an intake Queue). Each user can take a record from the queue. Because this option cannot be disabled after it is enabled, you should activate the Queues option only if it is likely that you will have to manage queues for this specific entity.

***Activate the Business Process Flows option only if it is required.***

Business Process Flows define a set of steps for users to follow to complete a particular business process. When enabled, system customizers have the ability to add a process flow control with a number of steps to the entity’s form. Because this option cannot be disabled, you should activate the Business Process Flows option only if a defined process is required for users to follow when working with that entity (example Lead🡪 Opportunity🡪 Quote🡪 Sales Order business process flow).

***Modify all messages that relate to a system entity that has been renamed.***

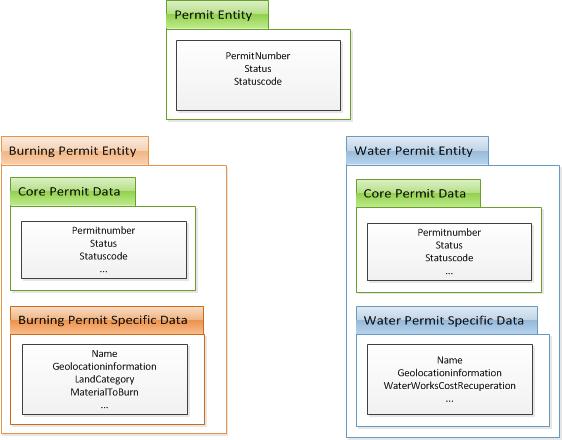
If a system entity is renamed (Account for example changed to Company), make sure all messages that relate to this entity are modified to reflect the new name. This is so that the user’s interface no longer refers to the previous name. Only the Display Name of an entity can be changed, schema names cannot be changed once the entity is created.

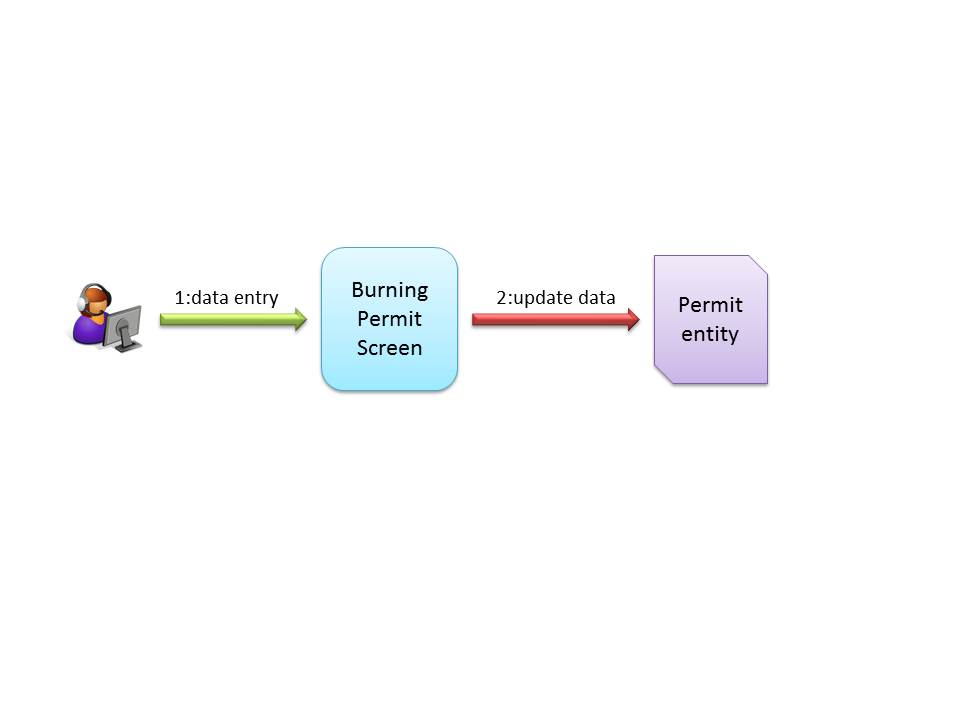


***Select the appropriate pattern for entities varying by category or type***

For simple forms, business rules can be use to show or hide attributes based on the form’s context (a selected category or type value for example). For more complex forms, it might be required to have different entities where some attributes are common.

In this scenario, it is suggested to duplicate (denormalize) the attributes into a common entity as CRM does not support inheritance between entities. This approach follows the “one entity per screen” requirement of the platform and provides a better reporting experience. Consider the following example:





On step 2, a business logic component will be triggered in order to copy/update the burning permit information into the “permit” entity. This business logic should be generic enough so that it could be used by all the different permits.

***Limit the number of entities that are enabled for offline use***

Carefully consider if an entity must be available for people while working offline. Each entity that you enable for offline capability directly affects the time required for people to synchronize data when they come back online. This is especially true for people with less powerful computers.

#### Attributes

***Apply a homogeneous naming schema to specific attributes.***

Applying a naming schema enables you to standardize the handling of specific attributes of a Microsoft Dynamics CRM entity:

* “Boolean” attributes are prefixed by “is” (for example, Contoso\_isManaged).
* “Lookup” attributes are suffixed by “Id” (for example, Contoso\_ecifId).
* “OptionSet” attributes are suffixed by “Code” (for example, Contoso\_contactTypeCode).
* “Currency” attributes are suffixed by “Amount” (for example, Contoso\_creditAmount).

***Define the requirement level of a field according to related business rules.***

The setting of the requirement level of an attribute to Business Recommended/Business Required is enforced only on the user interface.

If the requirement level of a field is set to Business Required, and you are importing data through the Microsoft Dynamics CRM import module, it is possible to leave the field blank. Hence, additional business layer validation is recommended if business requirement level must be enforced to maintain system referential integrity.

***Activate the Field Security profiles only when it is specifically required.***

Field Security profiles allow the user to manage the visibility of a specific attribute. This option adds a layer of security to the specified entity. Whenever this field is in a view, a form, or a web service request, the API will check to see whether the user is allowed to access the information. The excessive use of field-level security has negative performance impacts, with regard to the display of forms, views, and web services calls that relate to those fields.

***Activate Auditing only when it is required.***

Auditing enables Dynamics CRM to track changes to a particular field. When the field was changed, the user who performed the action, and the action type are logged. Enabling Auditing for a large number of fields has negative performance implications on the SQL database and its size. Care must be taken when defining which fields require to be audited for valid business reason. Avoid turning Auditing on by default for all fields.

***Prefix the label of the unused attributes of a system entity by using "Z\_".***

For example, if the attribute that has the Address 2: Street 1 label is not used, its value should be updated to Z\_Address 2: Street 1, so that the field will appear at the end of the columns that could be selected (for instance) for an advanced find.

***Select "no" for the Searchable option for unused attributes.***

This option enables you to use the attribute as a search criterion in an advanced find. To make a user’s interface clearer, this unused field should not appear in the possible search criteria.

#### Relationships

The definition of relationships is a very important step in the Microsoft Dynamics CRM data-model definition. When you are creating a relationship, it is important to correctly configure its Type of Behavior. Updating the parent record may or may not update the child records, depending on the relationship type of behavior.

The following categories of changes are defined:

* **Assign:** when the parent record is assigned to another user
* **Share:** when the parent record is shared with another user
* **Unshare:** when the parent record is unshared with another user
* **Reparent:** when the parent record’s parent is changed
* **Delete:** when the parent record is deleted
* **Merge:** when the parent record is merged

And the following cascade behaviors are available:

* **Cascade All**: cascade the change to all child records
* **Cascade Active**: cascade the change only to active child records
* **Cascade User-Owned**: cascade the change only to the child records owned by the same owner as the parent record
* **Cascade None**: do not cascade the change to the child records
* **Remove Link**: when the parent record is deleted, remove the link from child records to the parent record
* **Restrict**: the parent record cannot be deleted if there is a related child record

***Be careful when you define the relationship type while you are creating a new relationship.***

The choice of a relationship type can have an impact on the execution time of the cascade actions.

For example, a client has 500 child records, and the deletion relationship behavior is defined as Cascade All. If this client were to be deleted, the platform would delete the 500 child records before it deleted the client.

***Consider parent/child hierarchy in your data model.***

Because a child entity can have only one Parental relationship (i.e. a child has one parent), it is recommended to consider the parent/child hierarchy in the early stages of data modeling.

***Apply a naming convention for new relationships.***

The relationship name should refer to the name of the parent entity and the name of the child entity.

For example, a relationship between the account entity and the custom client entity should be named something like account\_Contoso\_client.

***Apply the appropriate cardinality.***

On an entity on which another entity is referenced through a Lookup attribute (for example, a client’s country), within the relationship definition on the client entity, choose the N to 1 cardinality.

On an entity that is used to reference child records (for example, accounts of a client), choose the 1 to N cardinality.

#### Connections

***Use connections, instead of native N-N relationships where possible.***

Native N-N relationships are used less and less. Connections are recommended to display relationships between records. The connections enable the user to define a connection role.

Create, delete, and update of connections between records is easier to work with via the SDK than N-N relationships.

***Do not customize the Connection entity; it is the same for every kind of connection.***

Because the Connection entity form is shared for every connection, it is better to avoid customizing it.

### User Interface

#### Forms

When you configure a CRM form, you should identify the pieces of information that are likely to be the most relevant. This information should be displayed by default when the form opens (default form). Other information can be displayed in other tabs, which will be collapsed by default. Knowing which pieces of information are the most used is crucial both to the optimization of the loading time of the form and to its clarity.

***Limit the number of fields within a form.***

Too many fields in a form can have a negative impact on its loading time.

***Limit the number of web resources that are involved within a form.***

To optimize performance, do not use too many web resources within a form.

***By default, define the tabs as collapsed.***

The loading time of a form is quickest when its tabs are collapsed by default. These tabs typically contain less used fields, and the user can expand the tab to view the fields in there.

***Limit the grid views that are displayed upon the opening of the form.***

While a form is loading, the application can load a maximum of four sub–grid views; the subsequent sub–grid views that are also visible on the form would require user action to load their contents. For obvious reasons, it is better to place grid views whose information is less important in a tab that is collapsed by default. The content of these views will be loaded only when the user expands the tab (on demand).

***If an iframe element is defined on the form, you should place it in a tab that is collapsed by default; you will be able to load it only by opening that tab.***

To optimize the loading time of the form, the iframes load must be effective only by user demand and not automatically upon the loading of the form.

***Keep mobile in mind***

The same form that is designed for the web client, is used to render the mobile client form as well. The rendering adapts to the form factor of the device it is rendering on, however, careful consideration must be taken when you’re laying out the form to be optimized for touch and mobility. Too many tabs on the form translate to significant horizontal scrolling on the mobile client.

***Keep number of steps and stages in Business Process flows low***

The process bar was designed to provide an easy way for users to follow an established process. It’s meant to encourage the right outcomes. It is not a wizard for data capture. It is recommended to use the process bar like a checklist to encourage the right practices and to enforce checkpoints at various stages.

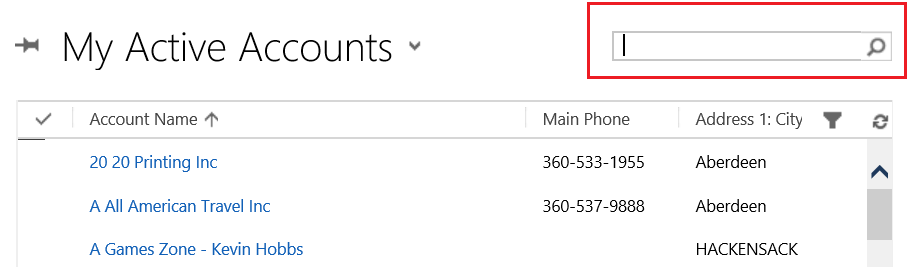
#### Views and Queries

***Reduce number of fields enabled for Field Level Security option in the most used system views.***

Try to limit the use of fields that are enabled for Field Level Security option in the most used views (for example, My Accounts, My Contacts, or All Accounts). This is because the display of these fields require a processing latency to define the visibility level of each particular field column in the view for the current user.

***Limit the number of Find Columns in the Quick Find View for an entity.***

These fields allow the user to search in the global view of the entity. The more fields specified in the Find Columns for the Quick Find view, the more columns the platform needs to perform the search against when the user types in a search keyword in the search box, and the longer the search operation takes.



***Apply filter criteria to the various views for an entity.***

Avoid views that return the entire dataset for a particular entity (for example Active Accounts, Active Contacts). Even though the result set is paged, the returned dataset is large and would require multiple pages to be loaded and managed by the platform. Consider what the business requirement for a particular view is (quarterly, by type, etc.) and try to limit the number of returned records.

***Apply conditions on unique value that is the same on all environments***

Record IDs are environment specific. Consider adding a custom field (like “CRMID”) and ensure the same value is used across environments for records that need to be referenced. For example, if you need to filter a view based on a lookup value (like a specific product), refer to a field in the record (like product’s code) instead of the record itself (which would internally refer to the record ID).

This rule applies to many components, including views, workflows, dashboards, charts, and also any source code like JavaScript, Plugins and Workflow custom activities.

You can reference directly an OptionSet entry because its value will be the same across environments.

### Processes

#### Workflows

From a performance perspective, is it better to create a single long workflow or is it better to have multiple child workflows and call them in one parent workflow? The child workflow approach achieves lower throughput, but it is more manageable if you frequently change your workflow definition. Compilation overhead is not a major concern because the workflow is compiled only during publishing. However, Microsoft Dynamics CRM incurs overhead when it starts each workflow instance. The overhead occurs when all entities that are used in the workflow are retrieved and the child workflow is started in a two-step process that includes a 'Workflow Expansion Task' and the actual workflow instance. Therefore, for maximum throughput, use a single long workflow.

***Check the trigger criteria of a workflow by starting a workflow through a test of the value of the trigger fields.***

The main risk in the configuration of a workflow is the creation of an endless loop. This can happen when the field that triggers the workflow is updated by the workflow.

***Package the workflows in a solution that is different from the solution that contains the data model.***

If a workflow is imported into an environment, it must be published before the changes can take effect. If the workflows are in the same solution as the data model, every solution import that affects the data model will necessarily involve publishing the workflows. If the workflows are packaged in a different solution, the workflow publishing step is required only when the workflows are updated.

***Minimize references to CRM records in a workflow definition.***

If a workflow refers to CRM records, the user must edit the workflow after it’s imported to an environment to update the reference of the CRM record. This is because the CRM record will have a different database identifier (GUID – Globally Unique Identifier) in the new environment. After the update is made the workflow must then be published.

***Avoid the use of long waiting workflows.***

Workflows provide the ability to wait for a certain event to occur in Dynamics CRM before they continue the execution (Wait Conditions). In this case, the workflow is in waiting state until it completes. A large number of waiting workflows might have a negative impact on the Async Services executing the workflow. Consider other scheduling patterns when there is a need to wait for a condition to be true before an action is taken.

***Cleanup workflow logs.***

Workflow execution is logged in Dynamics CRM and regular cleanup of the workflow logs improves the overall health of your system.

#### Dialogs

***Minimize the references to CRM records within a Dialog definition.***

If a Dialog refers to CRM records, a Dialog update—followed by a publish—will be required after the import of this Dialog from another environment.

It is much better to reference an identifier within the Dialog rather than the actual CRM record. (For example, if the Code field of the account is 123, launch an action, versus, if the account = 123 launch an action).

#### When to use plug-ins vs. workflow?

As a developer who is interested in extending or customizing Microsoft Dynamics CRM, you can choose from several methods to perform your tasks. In addition to adding client-side JavaScript code to a form or adding custom ASP.NET pages, you can write a plug-in or create a custom workflow by using the web interface that calls a custom workflow activity. How do you decide when to use a plug-in and when to use a workflow? The technology that you use depends on the task that you have to perform and who will author the customization.

For example, you must use a synchronous plug-in real-time workflow if you want to execute custom code immediately before or after the core platform operation executes and before the result of the operation is returned from the platform. You cannot use an asynchronous workflow or asynchronous plug-in in this situation because they are queued to execute after the core operation finishes executing. Therefore, you cannot predict when they will run. If you want to add custom functionality to Microsoft Dynamics CRM Online, workflows and plug-ins are supported, but custom workflow activities are not.

Evaluate these technologies and select the one that best suits your business objectives after you consider the deployment, performance, and maintenance concerns of your plug-in or workflow solution.

The following table summarizes the characteristics of plug-ins and workflows.

| Criteria | Plug-in | Workflow |
| --- | --- | --- |
| Execution before or after the core platform operation (Create, Update, Delete, and so on) | Executes immediately before or after the core operation (synchronous).  Can also be queued to execute after the core operation (asynchronous). | Asynchronous workflows are queued to execute after the core operation.  Real-time workflows have similar characteristics to plug-ins. |
| Performance impact on the server | Synchronous plug-ins can increase the platform response time because they are part of the main platform processing.  Asynchronous plug-ins have less impact on server response time because the code is run in a different process. | Asynchronous workflows have less impact on server response time because the code is run in a different process.  Real-time workflows have similar performance characteristics to sandboxed plug-ins. |
| Security restrictions | To register a plug-in with the platform requires a System Administrator or System Customizer security role and membership in the Deployment Administrator group. | Users can interactively create workflows in the web application.  However, to register a custom workflow activity, the deploying user must have the same security roles as those required for registering plug-ins. |
| Microsoft Dynamics CRM version (SKU) support | Supported in Microsoft Dynamics CRM Online when registered in the sandbox. May be supported in partner-hosted installations at the discretion of the partner. | Workflows are supported by all CRM deployments. Custom workflow activities are supported in the sandbox of Microsoft Dynamics CRM Online, and in or outside the sandbox for on-premises/IFD deployments. |
| Length of processing time | A plug-in registered for synchronous or asynchronous execution is restricted to complete its execution in a two-minute time limit. | Works well for either short or long processes. However, each activity in a workflow cannot take longer than two minutes to complete. |
| Works when the Dynamics CRM for Outlook client is offline | Both online and offline are supported. | Workflows do not execute when offline. |
| Process and data persistence | Plug-ins execute until they are completed. Plug-ins must be written to be stateless where no in-memory data is persisted. | Asynchronous workflows can be paused, postponed, canceled, and resumed through SDK calls or by the user through the web application. The state of the workflow is automatically saved before it is paused or postponed.  Real-time workflows cannot have any wait states. They must execute to completion just like plug-ins. |
| Impersonation | Plug-ins can perform data operations on behalf of another system user. | Asynchronous workflows cannot use impersonation, while real-time workflows can. Real-time workflows can execute either as the owner of the workflow or as the calling user. |
| Authoring | Software engineers or programmers can author plug-ins. | Anyone, including an end user, business analyst, or administrator can author workflows if they have the proper permissions. |

There is no significant performance impact on the server between an asynchronous plug-in and a workflow.

### Security Model

#### Business Units

***Create CRM business units according to the operational organization of the project, in terms of access rights to the data.***

Business Units in Dynamics CRM do not need to reflect an organization structure. Keep access rights and levels of access to data in mind when determining the Business Unit structure for your implementation. Think about the Business Unit structure very early on in your implementation.

#### Security Roles

Create a new role with those specific privileges and add the user to the new role if a user needs additional access levels or rights. A user's rights are the union of all the roles to which he or she has been assigned. Do not grant the original role privileges that are needed by only one or several members.

Use teams to create cross-functional groups so that specific objects can be shared with the team.

Train users who have sharing access rights to share the minimum information needed.

***Allocate the minimum required access privileges to each user.***

Allocate to each user the minimum required privileges, according to the business rules that are involved.

Elevation of privilege attacks occur when a user can assume the privileges of a trusted account, such as an owner or administrator. Always run under least-privileged user accounts and assign only needed permissions. Avoid using administrative or owner accounts for executing code. This limits the amount of damage that can occur if an attack succeeds. When performing tasks that require additional permissions, use procedure signing or impersonation only for the duration of the task.

***Assign the System Administrator role to a restricted number of users.***

Users who have the System Administrator role have very important access rights; it is important that the number of such users be limited, to limit the risk of improper handling of the system. Never remove this role.

***Do not delete Microsoft Dynamics CRM default security roles.***

If you do not want to use a default CRM security role, you can rename it by using “Z\_” as a prefix.

***Use the sharing option when sharing is required.***

If a user has access to a CRM record, but the privileges that are allocated to the security roles of this user do not allow the user to access the record, the owner of the record can share it with this user. Record sharing should be performed by using the minimum required privileges. Sharing should also be used sparingly, because it can greatly decrease performance; therefore, sharing should be the exception and not the rule.

***Allocate the System Customizer role to users who can modify the data model and the user interface.***

This role includes privileges that permit the user to modify the data model and the user interface of the application. However, the System Administrator role will still be necessary to deploy the plug-ins. Never remove this role.

***Take a layered approach to Security Roles***

When designing your Security Roles, grant access through a layering approach of the Security Roles. Each Security Role layer, grants the user elevated access to a particular area of Dynamics CRM. Consider the following example:

* All users require global read access to the Contact entity
* Only Sales staff require edit access to the Contact entity
* Only Sales Managers require edit access to the Opportunity entity

Define your layered Security Roles as follows:

* Base Role: grants user global level read on the Contact entity
* Salesperson Role: grants user edit access to the Contact entity
* Sales Manager Role: grants user edit access to the Opportunity entity

And assign Security Roles to users in layers:

* Basic users: assign them Base Role
* Sales staff: assign them Base Role and Salesperson Role
* Sales Managers: assign them Base Role, Salesperson Role and Sales Manager Role

#### Field Security Profiles

When specific fields are enabled for Field Level Security option, they belong to each Field Security profile of the application.

On the standard Field Security profile (System Administrator), all secured fields can be accessed in Read / Update / Create.

On each new security profile, all of the fields in the Field Security profile appear with a minimum allowed access (no access). The person who is configuring the security profile will be able to allow the minimum required access on the set of fields that are involved within the security of the field.

If a user has different security profiles, the maximum access that is available to that user applies. For example, if a user can read the Amount field through a Manager profile and can update that field through a Director profile, the user will be able to both read and update that field.

If a user is neither a System Administrator nor related to at least one security profile, the user will have no access to the fields in the Field Security profile.

***Assign Teams to Field Security Profiles***

The association of a team to a Field Security profile enables the team to inherit the access levels defined by the Field Security Profile, and by adding and removing users from the team, access can be granted or revoked. This is simpler to manage versus the scenario where security profiles are associated to individual users.

***Do not create a profile that specifies that some users have absolutely no access to secured fields.***

This is the default behavior; therefore, there is no value to define a profile for this purpose.

#### Teams

In Microsoft Dynamics CRM, teams can own the CRM records directly. For those teams to be able to own records, you will have to allocate to them one or more security role by applying the minimum required privileges. Like each user, each team is related to a business unit.

When users are added as team members, they keep all of the privileges that relate to their security roles, and they gain the privileges that are allocated to the security roles of the team. Because the security roles are cumulative, adding a user to a team can never restrict the privileges of that user, but it might allow the user to use new privileges.

***Allocate the minimum privileges to teams, according to the visibility needs of the CRM records.***

To verify the safety of the application data, you should not give privileges to a team unless that privilege is necessary for the team.

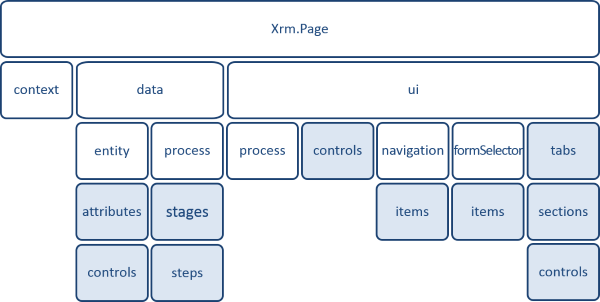
***Limit the number of teams and team memberships in your application***

The teams a specific user belongs to, and what access each team provides the user is initialized when the user accesses Dynamics CRM. It is recommended to keep the number of teams, team memberships, and team sharing to a minimum.

## Extending Microsoft Dynamics CRM

### Client Side Scripting – JavaScript

#### Xrm.Page Namespace



The **Xrm.Page** namespace contains three objects:

* **Context**—provides the methods by which the user can access specific information that relates to the organization:
  + URL of the Microsoft Dynamics CRM server
  + Language code of the connected Microsoft Dynamics CRM user (for example, 1033: English or 1036: French)
  + Security roles of the current user
* **Data**—provides the methods by which the user can access the data of the entity:
  + Attribute value
  + Kind of attribute
  + Options list (for optionsets)
* **UI**—provides the handling methods of the user interface:
  + Activation and deactivation of the read-only mode on a field
  + Management of field visibility
  + Access to elements of the user interface (for example, tabs and sections)

***Use Visual Studio to create and edit JavaScript files.***

To optimize JavaScript development, we recommend the use of Visual Studio. By using this tool, the user can utilize colored functions and IntelliSense (code completion). The Developer Toolkit for Microsoft Dynamics CRM is a powerful set of tools for Visual Studio that streamline the development and deployment of custom Dynamics CRM solutions. See section 3.4 Developer Toolkit for Microsoft Dynamics CRM for more details.

***Whenever possible, group common functions in a JavaScript library.***

When a function is reused in several forms, it is better to group this function in a common web resource that will be named according to its use (for example, CONTOSO.Common.js).

***Define a namespace for JavaScript web resources.***

Example:

if (typeof CONTOSO == "undefined") {

     CONTOSO = { \_\_namespace:true };

}

***Verify that there is an attribute on the form before accessing it.***

Before the user can invoke or update an attribute in JavaScript, the user must first verify that the field is on the form:

if (typeof CONTOSO == "undefined") {

     CONTOSO = { \_\_namespace:true };

}

CONTOSO.Common = {

    getValue: function () {

        var attribute = Xrm.Page.getAttribute('Contoso\_something');

        if (attribute == null) {

            throw new Error("This field Contoso\_something is not defined on the form.");

        }

        return attribute.getValue();

    }

};

***Do not modify the DOM of CRM forms.***

Refrain from accessing or modifying the DOM elements of Dynamics CRM forms. Always use the XRM framework to access and work with Dynamics CRM in client side scripts. This is because the internal implementation of the DOM can change, and your code will need to be re-factored.

When extending CRM using JavaScript, jQuery can be used with web resources (HTML components including JavaScript) but it should be avoided for form scripts and ribbon commands. While jQuery simplifies cross-browser development, it manipulates the DOM directly. Restrict your scripts to use Xrm.Page and Xrm.Utility, with $.ajax when needed.

Details: <https://msdn.microsoft.com/en-us/library/hh771584(v=crm.6).aspx>

***Review the security of client customizations.***

* Adhere to the requirement that your custom Web pages be installed on a Website separate from Microsoft Dynamics CRM. Set the trust level for your site appropriately, depending on your confidence level in the security of your code. Doing so mitigates the threat from cross-site scripting and other threats.
* Avoid use of ActiveX controls because they have known security problems.
* Do not use JScript to perform complicated functions. Use plug-ins instead.
* Remember that JScript can be bypassed by users and validations should be performed on the server for integrity.
* Any data validation performed using JavaScript will be ignored by any application or service using directly the SDK or CRM web services.
* While form programming provides capabilities to hide or disable form fields so that users may not be able to view or update some entity data, it does not represent a complete solution for enforcing security requirements. A user can see entity data that is not visible on the form by using a variety of methods, such as **Advanced Find**.
* Remember that your JScript code is running under a high-trust level and should be used primarily for functions such as data validation and completion.
* Do not call remote Websites or remote procedures from JScript.

***Manage the exceptions properly during the execution of a CRM plug-in.***

Refer to the recommendation with the same name under Plug-ins. Of course, the JavaScript code would need an entry point (like an Action in CRM and/or using Visual Studio Application Insights on Azure) to store the exception for the administrators.

***Manage environment-specific configurations.***

Refer to the recommendation with the same name under Plug-ins.

#### Web Resources

***Use a naming convention for web resource names, to classify and easily identify the kind of resource and its use within the application.***

Define the web resource names according to a convention such as the following:

• Contoso/js/CONTOSO.Client.js for JavaScript functions that relate to a client form

• Contoso/html/ClientView360.htm for an HTML page that is deployed on the client record

• Contoso/img/Bouton1.png for a PNG (graphics) file

• Contoso/css/ClientView360.css for the style sheet of the HTML file

***Import only supported web resources types.***

The list of supported web resource types is available at the following location:

<https://msdn.microsoft.com/en-us/library/gg309473(v=crm.6).aspx>

***Choose HTML, JavaScript, and OData applications over Silverlight and external applications.***

The development of an extension of the user interface (an iframe element on an entity form, or a specific entry of the site map) will be performed in HTML or JavaScript. This method enables the user to create light weight applications—for example, application of a quick contact creation—that reference the representational state transfer (REST) services (OData) that are to be provided by Microsoft Dynamics CRM.

Because Microsoft Dynamics CRM does not allow users to modify the DOM of the application forms, the integration of extended applications in Microsoft Dynamics CRM has to be performed by using Iframe. Extended applications should at least pass the HTML 4.1 standard.

Avoid use of ActiveX controls because they have known security problems.

Be aware of the limitations of client scripting. More information: [Write code for Microsoft Dynamics CRM forms](https://msdn.microsoft.com/en-us/library/gg328261.aspx).

#### Form and Fields Events

***Confirm that the data is updated upon the launch of an OnSave event.***

The software development kit (SDK) specifies that the OnSave event is created as soon as the user clicks Save. It does not check whether or not the entity has effectively been updated. A test to check for updates (if Xrm.Page.data.entity.getIsDirty()) will allow the user to avoid unnecessary processing if the entity has not been updated.

***Avoid synchronous heavy processing (such as a web service call or a complex JavaScript calculation) that relates to an OnLoad event.***

Because the OnLoad event happens each time that a form is opened, if the JavaScript process is long, the loading time of the page will be increased as long as the process runs. This can have a negative impact on the connected user.

***Confirm that the OnSave event has not already been cancelled by another handler.***

If several JavaScript functions are registered on the form’s OnSave event, use isDefaultPrevented() to check whether the Save operation has been cancelled before applying your custom logic:

if (typeof CONTOSO == "undefined") {

     CONTOSO = { \_\_namespace:true };

}

 CONTOSO.Common = {

    onSave: function(ctx) {

        ctx.getEventArgs().getSaveMode();

    },

    onSave2: function(ctx) {

        if (ctx.getEventArgs().isDefaultPrevented()) {

            return;

        }

        // Launches the operation if the event happens

    },

};

***Fill in the URL of the iframe in the TabStateChanged event of the section.***

The iframe element must be loaded in the TabStateChanged event of the section that contains the iframe element.

***Use the getEventSource method to define generic functions.***

When you want to define a generic function (such as a number format in a form), it is recommended to use the getEventSource method on the ExecutionContext object that is transmitted to this method, because this method prevents the user from having to define a specific attribute in the function.

The CONTOSO.Common.formatnumber method can be used for any string field:

if (typeof CONTOSO == "undefined") {

     CONTOSO = { \_\_namespace:true };

}

CONTOSO.Common = {

    formatNumber: function (eContext) {

        var attribute = eContext.getEventSource();

        var newValue = "+1" + attribute.getValue();

        attribute.setValue(newValue);

    }

};

#### SOAP and REST Endpoints

***Use Web API (REST) instead of CRM 2011 Endpoint (SOAP)***

Beginning with the CRM 2016 (8.0) release, a new OData 4 endpoint was added to the CRM application stack. This endpoint is known as the Web API. The Web API provides a development experience that can be used across a wide variety of programming languages, platforms, and devices. The Web API is intended to replace the CRM 2011 endpoint, also known as the SOAP endpoint, for accessing the CRM Organization web service and other provided web services.

As of the release of Microsoft Dynamics CRM Online 2016 Update 1 and Microsoft Dynamics CRM 2016 Service Pack 1 (8.1.0), the CRM 2011 endpoint has been deprecated. The 2011 endpoint will be removed sometime after CRM version 9. We plan to provide updates to the CRM SDK assemblies and tools over the next several minor releases, retargeting them to use the Web API instead of the 2011 endpoint.

We strongly encourage you (developers) to use the Web API for accessing the CRM web services when you write new application code. You must update existing applications to use the CRM Web API for accessing the CRM web services between now and the removal of the 2011 endpoint. When you develop plug-ins and custom workflow activities, continue to link those projects to the CRM SDK assemblies. The CRM SDK assemblies will be updated to use the Web API. This update will be fully transparent to you and any code written using the SDK itself will be supported.

Additional details will be published in future SDK releases. For more information about the Web API, [Use the Microsoft Dynamics CRM Web API](https://msdn.microsoft.com/en-us/library/mt593051.aspx)

*Source:* [*Important changes coming in future releases of Microsoft Dynamics CRM*](https://msdn.microsoft.com/en-us/library/dn281891.aspx)

***Make asynchronous web service calls.***

If a web service call is performed synchronously, each web service that is generated from the user interface can block the user interface as long as it is waiting for the CRM server response.

***Filter data that is requested during a web service call.***

The REST and Simple Object Access Protocol (SOAP) calls allow the user to select the information that he or she wants to collect during a web service call. For the best performance, each web service call should collect only the minimum required information from the CRM server.

For example, the /AccountSet(guid'[GUID]')?$select=AccountNumber URL provides the account number that relates to the account with the specified GUID.

***Access elements that relate to the navigation links that are provided by the REST protocol.***

For example, if you want to collect the opportunities of a specific account, the URL that allows the user to retrieve them is the following:

/AccountSet(guid'[GUID]')/opportunity\_customer\_accounts

***Use community adopted tools to generate complex REST queries.***

Microsoft Dynamics CRM external tools enable the user to generate the URL that relates to a particular call.

Dynamics XRM Tools (see http://www.xrmtoolbox.com/) allows the user to generate complex OData queries easily.

#### Interaction with External Web Applications

Best practices for developing customizations for the CRM web application and Microsoft Dynamics CRM for Microsoft Office Outlook include the following:

* Use web resources instead of pages that require server-side processing whenever possible. If your requirements can only be achieved by using server-side processing, adhere to the requirement that your custom webpages are installed in a separate website from Microsoft Dynamics CRM. Set the trust level for your site appropriately, depending on your confidence level in the security of your code. This reduces the threat from cross-site scripting and other threats.
* For improved security, make sure that your separate website runs on a different account from Microsoft Dynamics CRM. This account should have the minimum access possible and one that does not have direct access to the Microsoft databases. You can use a complex password that doesn’t expire because no person signs in to this account – only in to your application.
* Avoid use of ActiveX controls because they have known security problems.
* Be aware of the limitations of client scripting. More information: [Write code for Microsoft Dynamics CRM forms](https://msdn.microsoft.com/en-us/library/gg328261(v=crm.6).aspx)
* Use plug-ins to apply business logic regardless of how the data changes are made.
* Always use a modal confirmation dialog box when you delete records or apply sensitive changes, such as adding a new user to a security role. This helps prevent techniques such as click-jacking or UI redressing where a malicious developer may embed your page in a seemingly innocuous page to trick a user into performing actions that may compromise security or perform unwanted actions on data.

Security best practices for your website include the following:

* Don’t use anonymous access.
* Use integrated Windows authentication, NTLM, or Basic authentication over Secure Sockets Layer (SSL).
* Use SSL to avoid sending unencrypted data over the network if your website is on a different computer than Microsoft Dynamics CRM.

For more information, see the following:

* [Overview of Web Application Security Threats](https://msdn.microsoft.com/en-us/library/f13d73y6.aspx)
* [Microsoft Anti-Cross Site Scripting Library V4.2](https://www.microsoft.com/en-us/download/details.aspx?id=28589)
* [ASP.NET Web Application Security](https://msdn.microsoft.com/en-us/library/330a99hc.aspx)
* [Introduction to Web Application Security](https://msdn.microsoft.com/en-us/library/aa711426(VS.71).aspx)

It is possible to enhance the Dynamics CRM standard functions by using ASP.NET applications that interact with CRM web services.

In the following sections, you will find a set of recommended practices that relate to this extensibility function.

***Always deploy an ASP.NET application on a website other than the CRM website.***

If an ASP.NET application that is querying CRM is used within a project framework, the user must deploy this application on a website other than the CRM website.

***Define an iframe element on an entity form if the external web application uses the context information of the opened record.***

Because Microsoft Dynamics CRM does not allow the user to modify the DOM of the application forms, the user might have to create extended application integrations in Microsoft Dynamics CRM by using Iframe.

An Iframe definition in a form enables the user to transmit background information to the page that is opened in the Iframe. The background information includes:

* The unique identifier of the opened record.
* The name of the CRM organization.
* The name of the entity.
* The number that relates to the entity type.
* The language of the connected user.
* The basic language of the organization.

***To reduce the loading time of a form, load the content of an Iframe only upon user demand.***

Usually, the Iframe is set in a tab that is collapsed by default. When the user opens this tab, the URL of the Iframe is filled in, and the external web application is loaded.

***Define an entry in the site map if the web application does not have to display background information that relates to the record.***

The Site Map can be configured to add links to external applications. An example could be a link in the Site Map called ECIF Search, that would launch the ECIF UI application to search for a particular client.

***Connection Strings must be encrypted***

Use Windows integrated security (App Pool Identity account for example)

Do not use explicit usernames and passwords stored in plain-text.

Implement secured connection strings and configuration sections that contain sensitive information using Protected Configuration. Use built-in ASP.NET IIS Registration Tool (Aspnet\_regiis.exe).

Please refer to: [Encrypting and Decrypting Configuration Sections](https://msdn.microsoft.com/en-us/library/zhhddkxy.aspx)

Note: If you are planning on using the same encrypted configuration file on multiple servers, such as a Web farm, only the RsaProtectedConfigurationProvider enables you to export the encryption keys and import them on another server.

***Apply secure development practice***

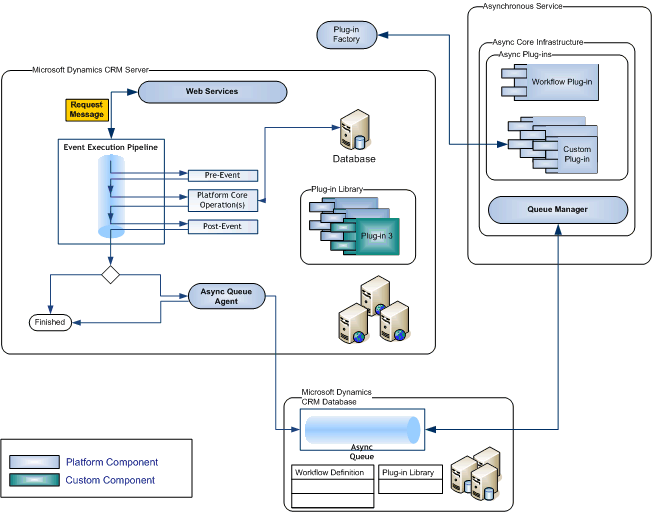
* Do not modify the Microsoft Dynamics CRM database by any means other than using the SDK because this bypasses the Microsoft Dynamics CRM security model.
* Plug-ins are running in an administrator's context – you should know that this code may access information that the logged-on user does not have access to.
* For workflow assemblies, and plug-ins, avoid writing code that takes a long time to execute. It is important that plug-in code that is registered to execute synchronously returns as quickly as possible.
* If you are replicating Microsoft Dynamics CRM data in your own data store, you are responsible for the security of the data. If you use a plug-in to transmit the data, make sure that you register the plug-in to execute after the core platform operation. Security privilege checks for the logged-on user occur during the core platform operation.
* Data that comes out of Microsoft Dynamics CRM might not be safe for rendering. Data may have been injected with HTML tags that are not secure.
* Adhere to the requirement of not accessing Microsoft Dynamics CRM databases directly through SQL Server Enterprise Manager. Bypassing the SDK can open you up to SQL injection threats.
* For Internet facing deployments, remember that your solution is only as secure as the weakest link. After your application is exposed to the Internet, it is open to security threats.
* Use only languages that produce managed code for the best security from buffer overruns, exceptions, and so on.

For more information about security, see the following topics:

* [.NET Framework Developer's Guide, Securing Applications](http://msdn2.microsoft.com/en-us/library/aa720576(VS.71).aspx)
* [Secure Coding Guidelines for the .NET Framework](https://msdn.microsoft.com/en-us/library/8a3x2b7f.aspx)
* [ASP.NET Security](https://msdn.microsoft.com/en-us/library/91f66yxt.aspx)

### Plug-ins

#### Event Execution Pipeline



The event pipeline is divided into multiple stages, of which 3 are available to register custom developed plug-ins. Multiple plug-ins that are registered in each stage can be further ordered (ranked) within that stage during plug-in registration.

| Event | Stage name | Stage number | Description |
| --- | --- | --- | --- |
| Pre-Event | Pre-validation | 10 | Stage in the pipeline for plug-ins that are to execute before the main system operation. Plug-ins registered in this stage may execute outside the database transaction. |
| Pre-Event | Pre-operation | 20 | Stage in the pipeline for plug-ins that are to execute before the main system operation. Plug-ins registered in this stage are executed within the database transaction. |
| Platform Core Operation | MainOperation | 30 | In-transaction main operation of the system, such as create, update, delete, and so on. No custom plug-ins can be registered in this stage. For internal use only. |
| Post-Event | Post-operation | 40 | Stage in the pipeline for plug-ins which are to execute after the main operation. Plug-ins registered in this stage are executed within the database transaction. |

***Important White Paper: “Scalable Customization Design in Microsoft Dynamics CRM”***

Microsoft Dynamics CRM is designed to protect itself and its users from long running activities that could affect both the response times for the user making a request and the stability and responsiveness of the system for other users.

A challenge faced by some customers implementing Dynamics CRM solutions are errors thrown by the platform or the underlying Microsoft SQL Server database when these protective measures take effect. This is often interpreted as the platform not being able to scale or incorrectly terminating or throttling requests to the system.

The white paper (link below) is based on experiences investigating and addressing the true underlying causes of the majority of these types of challenges. It describes how the platform protects itself from the impact of these requests imposed on the system, as well as explains why this behavior is most often the result of custom implementations not understanding the impact on blocking and transaction usage within the platform.

It also describes how optimizing a custom implementation to avoid these types of behaviors will not only avoid platform errors, but also enable better performance and end user experiences as a result. It provides good design practices and identifies common errors to avoid.

White Paper: [Microsoft Dynamics CRM 2015 and Microsoft Dynamics CRM 2016 Performance and Scalability Documentation](https://www.microsoft.com/en-us/download/details.aspx?id=45905)

***Allow the system to create GUIDs***

Allow the system to automatically assign the GUID (Id) for you instead of manually creating it yourself. This suggestion allows Microsoft Dynamics CRM to take advantage of sequential GUIDs, which provide better SQL performance. The following sample code shows how to call the [Create](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.create.aspx) method to obtain a system-assigned GUID.

// Instantiate an account object.

Account account = new Account { Name = "Fourth Coffee" };

// Create an account record named Fourth Coffee and retrieve the GUID.

accountId = serviceProxy.Create(account);

***Consider supported Microsoft Dynamics CRM messages and entities in the development of a CRM plug‑in.***

The list of messages and entities that are supported in the development of plug-ins is available in Excel format in the SDK\Tools\Message-entity support for plug-ins.xlsx file.

The CrmService common methods are faster than using the CrmService.Execute method with the corresponding message. For example, use Create Method instead of using Create Message.

***Do not create or update CRM metadata in a CRM plug-in.***

The creation and update of metadata are not supported by Microsoft Dynamics CRM plug-ins.

***Restrict the execution time of a CRM plug-in.***

The execution time of CRM plug-ins (sandbox isolation) is limited to 2 minutes, by default; if the time goes over this limit, an exception is raised by the CRM execution pipeline.

***Limit the plug-in recursion processes.***

By default, the recursiveness level of CRM plug-in processes is restricted to 7; above this limit, an exception is raised by the CRM execution pipeline.

***Consider the stage and order of all plugins holistically.***

The stage and order at which plugins run is critical for correct application behavior. Always consider the order of the other registered plugins, and identify where in the pipeline your logic must run.

***Do not use static or class level variables.***

The same plugin instance is re-used by the Dynamics CRM server to handle multiple instances of the same event for which the plugin is registered. Using class or static variables could result in unexpected behavior due to the shared plugin object between runs.

***Manage environment-specific configurations.***

Hard coding values like email addresses, URLs and other variables create maintenance issues if – more likely when – the values are changed. Also if the coder types the value incorrectly, the only way to find the error is for the code with the error to be executed. It is a possibility that the code containing the error will get past testing.

For those variables and any configuration that varies per environment, compile them in a central repository. In CRM, this is often using a custom Options entity (with key/value fields) and a common class helper that caches the values.

***Documentation***

Keep the building blocks (methods) small and clear. A method should do a single, well-defined job. It should therefore be easy to name a method.

Put comments in your code, telling others what the code is supposed to do and what the parameters are used for.

Recommendation: Microsoft recommends the following guidelines to make comments clearer and more useful:

* Document the why, not the how or what. The code will show the how and what (assuming method and variable names are properly named – if a method must be named Apply6, ensure a complete description is added).
* Purge the code base of low value comments or dead code.
* Prefer documentation using the [XML Documentation](https://msdn.microsoft.com/en-us/library/b2s063f7.aspx) syntax in C# and [Comment-based help syntax](https://technet.microsoft.com/en-us/magazine/hh500719.aspx) in PowerShell.

C# Coding Conventions: <https://msdn.microsoft.com/en-us/library/ff926074.aspx>

***Limit duplicated code.***

Code clones are separate fragments of code that are very similar. They are a common phenomenon in an application that has been under development for some time. Clones make it hard to change your application because you have to find and update more than one fragment.

Finding clones is typically useful in the following cases:

* When updating existing code. When you are fixing a bug, or responding to changes in requirements, you usually start by finding the location in the code that you need to change. Before you make the change, search for clones of that code segment. If clones are discovered:

1. Consider whether you need to make the same change to each clone.
2. Consider also whether this is a good opportunity to refactor the cloned code into a shared method or class.

* Architectural cleanup. Towards the end of every iteration, use Analyze Solution for Code Clones on the Analyze menu.
* When you create code. When you have written new code, use the tool to find similar code that already existed.

Note: the tool available with Visual Studio ignores some file types (like aspx) where the amount of code should be limited.

Consider:

* Moving code in a Utility or Helper class
* Linking files using Visual Studio

***Use static code analysis tools***

Static Analysis employs tools to automatically analyze source and binaries. These techniques can be used to enforce coding standards, find common errors and failures to follow best practice. Many of these tools can be extended to implement custom rules to enforce your own policies. Like other quality control processes, static analysis should be performed on a regular basis, ideally as part of a continuous integration process.

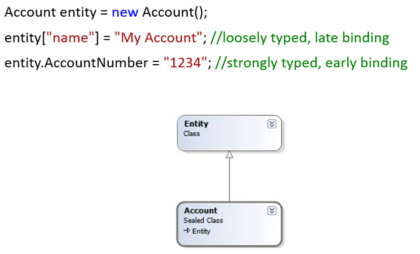
Enforcing a resolution (fix or valid reason to suppress warning) of every result in the “Microsoft Managed Recommended Rules” set should be highly considered (or at least “Microsoft Managed Minimum Rules” as a minimum requirement).

***Prioritize early bound entity classes with the code generation tool***

The CRM SDK provides two approach to binding data in code.

Early binding is implemented by generating strongly typed entity objects that include the schema of the entity. This facilitates development by exposing the schema through the object model, as well as enabling compile time errors to reduce run-time defects. The generated entity types need to be regenerated when the CRM schema changes and is therefore very static.

Late binding is implemented by using a generic Entity type. The Entity type has un-typed dictionaries to transfer data to and from CRM. This approach is more dynamic and flexible, since there is no generated code. It allows for reusable plugins that can accommodate a wide variety of scenarios. Since the data is stored in un-typed variables, there is significantly less compile time error handling and a higher risk of run-time defects.



The CrmSvcUtil.exe tool creates a Microsoft Visual C# or Microsoft Visual Basic .NET output file that contains strongly-typed classes for entities in your organization. This includes custom entities and attributes. This output file contains one class for each entity, providing early binding and IntelliSense support in Microsoft Visual Studio to aid you as you write custom code. The generated classes are partial classes that can be extended with custom business logic in separate files.

Use this method when you want to have the compiler check type references at compile time. Also, attributes and relationships are included in the generated class, so they have IntelliSense support for entity, attribute, and relationship names. Using strong types is generally the preferred method and is used in the majority of samples in the SDK documentation.

There are some downsides to this approach:

* Classes must be regenerated each time entities are customized to take advantage of schema changes.
* Serialization costs increase as the entities are converted to late bound types during transmission over the network

Use the [Entity](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.entity(v=crm.6).aspx) class when your code must work on entities and attributes that aren’t known at the time the code is written. In addition, if your custom code works with thousands of entity records, use of the [Entity](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.entity(v=crm.6).aspx) class results in slightly better performance than the early-bound entity types. However, this flexibility has a disadvantage because you cannot verify entity and attribute names at compile time. If your entities are already defined at code time and slight performance degradation is acceptable, you should use the early-bound types that you can generate by using the CrmSvcUtil tool.

This tool (CrmSvcUtil.exe) is in the SDK package in the SDK\Bin folder.

Example 1: GenerateEntities.bat

sdk\crmsvcutil.exe /l:CS /url:http://<*dev*>/XRMServices/2011/Organization.svc /o:..\GeneratedCode.cs /n:<namespace>.Entities /serviceContextName:XrmServiceContext

pause

Example 2: GenerateOptionSets.bat

sdk\CrmSvcUtil.exe ^

/codewriterfilter:"Microsoft.Crm.Sdk.Samples.FilteringService, GeneratePicklistEnums" ^

/codecustomization:"Microsoft.Crm.Sdk.Samples.CodeCustomizationService, GeneratePicklistEnums" ^

/namingservice:"Microsoft.Crm.Sdk.Samples.NamingService, GeneratePicklistEnums" ^

/url:http://<*dev*>/XRMServices/2011/Organization.svc ^

/out:OptionSets.cs

pause

More information: [Use the early bound entity classes in code](https://msdn.microsoft.com/en-us/library/gg328210(v=crm.6).aspx)

#### Dynamic Link Library (DLL) Referencing

***Limit the number of external dynamic link libraries (DLLs) that are referenced in a CRM plug-in.***

External DLLs that are referenced by CRM plug-ins should be limited to:

* CRM SDK DLLs.
* .NET framework DLLs that are compatible with the isolation level of the plug-ins.
* DLLs that are signed with a common technical basis and predeployed in the GAC with a standard isolation level. (Not applicable for CRM Online or if planning to eventually use CRM Online)

***Do not reference external DLLs that are deployed on the file system from the CRM plug-in.***

This kind of referencing is not compatible with database deployment of CRM plug-ins.

Having several external DLLs referenced in one plug-in is not compatible with either a sandboxed or an online solution.

***Do not merge several DLLs to enable them to be referenced by one CRM plug-in.***

Referencing that is based on the use of the ILMerge tool to merge many external DLLs into one Microsoft Dynamics CRM plug-in is too complex for the deployment and configuration of the Microsoft Dynamics CRM plug-ins. If issues arise, it will be more complex to debug the plug-in if the DLLs are merged.

***Favor the sharing of the source code among CRM plug-ins, to reuse common code among several plug-ins.***

This share mechanism is compatible with a database deployment of CRM plug-ins.

If you have to reference items in another project, you should use the AddAsLink function to refer to those items in your plug-in. This allows you to not have to duplicate the items and makes it easier to maintain if there are required updates.

#### IPlugin Interface

***Implement the IPlugin interface in a stateless way in a CRM plug-in.***

The Microsoft Dynamics CRM execution pipeline simultaneously carries out the same Microsoft Dynamics CRM plug-in instance on several concurrent threads for the occurrences of different events. Therefore, to avoid concurrent access to variables, member variables should be defined in the scopes of methods and never in the scope of their related parent class in the IPlugin interface implementation. If you use variables within a class, the code will no longer be thread-safe.

***Use Secure and Unsecure configuration to pass parameters to the plugin.***

When registering the plugin, configuration parameters can be passed through the Plugin Registration Tool to the plugin’s constructor, and it can then be accessed only in read-only mode in the Execute method of the CRM plug-in.

***Manage the exceptions properly during the execution of a CRM plug-in.***

Never let the user experience a runtime error. Always foresee such situations, and take appropriate action to either manage the situation programmatically or let the user know that an exception has occurred. Log the exception and the actions that can be taken to fix the problems.

To manage exceptions in CRM plug-ins properly:

* Intercept execution exceptions in a CRM plug-in by using a try {…} catch(…) syntax.
* Define a series of gradual interwoven catch () blocks to intercept exceptions from the most specific to the less specific.
* Trace the exception condition in the CRM API trace manager.
* Manage the exception condition specifically on a case-by-case basis.
* Raise the following exception when the exception condition cannot be caught: InvalidPluginExecutionException (String, Exception).

Always throw an **InvalidPlugInExecutionException. For custom workflow activities, this error will be shown in the workflow instance form. For plugins, this error will be shown directly to the user in a dialog.**

It is a good practice to store exceptions with execution details (context, stack trace, etc.). Various implementations are possible and the following list illustrates some topics to consider:

* Save details in a custom entity in CRM. This should be performed using an async Action in order to be outside of the executing transaction (which will be rolled back due to the exception thrown). While this option allows easy access to read the information for CRM administrators, it is not a usual practice for system administrators and monitoring tools. A purge system should also be considered.
* For on premises installation for plugins, exceptions could be stored in the Windows’ Event log which is a common storage location for system administrators and monitoring tools. When plugins are running inside the Sandbox processing service, access to the event log isn’t allowed but calling a custom web service to delegate the operation is possible despite it adds dependencies in the overall architecture of the solution. It is important to clearly state which organization created an entry in the event log (especially for installations with multiple organizations and/or if the custom web service writes to a central event log).
* In addition to storing the exception details on the servers, it should be considered to generate a unique identification number that can be shown to the end-user (in the message passed to InvalidPlugInExecutionException) that will also be saved in the exception details and in the CRM Trace log. This number would simplify research by the support them.
* Changing CRM’s Trace output level can be useful for debugging but can impact performance in a significant way, especially in production. One practice is to extend LocalPluginContext.Trace, that insert messages in the CRM TracingService, with a temporary collection (List<string>), that is specific to the plugin’s execution context and discarded once completed. When an exception occurs, the collection is appended to the exception log.

The strategy depends on the hosting location (especially for production servers), the level of details needed, who will need access to those details and how they can access it.

* More details: [Handle exceptions in your code (common exceptions and faults)](https://msdn.microsoft.com/en-us/library/gg327884(v=crm.6).aspx)

Consider integrating Dynamics CRM with [**Visual Studio Application Insights**](https://azure.microsoft.com/en-us/services/application-insights/) on Azure to proactively detect issues through email and webhook alerts; easily diagnose exceptions and web app performance issues; perform root cause analysis with ad-hoc queries and full-text search; and, integrate with DevOps processes using Visual Studio Team Services. This service can be integrated with web applications (hosted on Azure, other cloud services, or on-premises servers), including CRM in plug-ins, custom activities in workflows, and also client side scripting (JavaScript on forms).

* [Walkthrough: Enabling Telemetry for Microsoft Dynamics CRM Online using Application Insights](https://azure.microsoft.com/en-us/documentation/articles/app-insights-sample-mscrm/)

Once an exception is thrown, part of the information it carries is the stack trace. The stack trace is a list of the method call hierarchy that starts with the method that throws the exception and ends with the method that catches the exception. If an exception is re-thrown by specifying the exception in the throw statement, the stack trace is restarted at the current method and the list of method calls between the original method that threw the exception and the current method is lost. To keep the original stack trace information with the exception, use the throw statement without specifying the exception.

Bad example:

catch (Exception ex)

{  
 //do something with Ex or remove the local declaration for the catch block

throw ex;

}

Good example (would be better if not catching a general exception type):

catch (Exception ex)

{  
 //do something with Ex or remove the local declaration for the catch block

throw;

}

* Reference: [CA2200: Rethrow to preserve stack details](https://msdn.microsoft.com/query/dev12.query?appId=Dev12IDEF1&l=EN-US&k=k(CA2200);k(TargetFrameworkMoniker-.NETFramework,Version%3Dv4.0)&rd=true)

***Limit the use SharedVariables collection to share data among CRM plug-ins.***

The existing mechanism of data sharing among different CRM plugin instances based on the SharedVariables collection, produces tightly coupled Microsoft Dynamics CRM plug-ins that can damage the scalability of the solution. Use caution when using SharedVariables to pass information between plugins.

***Define a naming convention for the classes of a CRM plug-in.***

Applying a naming convention enables the user to standardize the implementation of CRM plug-ins—for example:

* CONTOSO.<Entity>.<Stage>.<Function>Plugin for a Microsoft Dynamics CRM plug-in that supports the <Function> functionality for entity <Entity> and is registered in the <Stage> stage. For example, CONTOSO.Contact.Pre.GenerateAutoNumberPlugin is a plugin that is registered on the Contact entity, in the Pre operation stage, and generates an auto number.

***Validate the Target input parameter***

Validating the presence of a valid Target input parameters, of the expected type (is Entity or EntityReference) and for the expected LogicalName (either by validating the string in LogicalName or using ToEntity<entity> using the generated classes based on the actual schema.

Exemple (could be transformed as a Target property in the LocalPluginContext class – see next recommendation):

// The InputParameters collection contains all the data passed in the message request.

if(context.InputParameters.Contains("Target") &&

context.InputParameters["Target"] is Entity)

{

// Obtain the target entity from the input parameters.

Entity entity = (Entity)context.InputParameters["Target"];

// Verify that the target entity represents an account.

// If not, this plug-in was not registered correctly.

if (entity.LogicalName != "account")

{

return;

}

***Implement a base class for CRM plug-ins.***

Implementing a base class for CRM plug-ins enables the user to set common behaviors for them—for example:

* Managing configuration settings that define the shared state of CRM plug-ins.
* Managing execution exceptions within CRM plug-ins.
* Validating the execution context of CRM plug-ins.
* Validating the presence of a valid Target input parameters (see above).

***Trace using the Plugin Tracing Log***

Use Plugin Tracing Service for debugging purposes. Under Select, System, Administration, System Settings (Customization), change the **Enable logging to plug-in tracing log** to the appropriate level for the current environment. See [Debug a plug-in](https://msdn.microsoft.com/en-us/library/gg328574.aspx) for details.

ITracingService tracingService =

(ITracingService)serviceProvider.GetService(typeof(ITracingService));

tracingService.Trace("Plugin Message " + context.MessageName.ToUpperInvariant());

***Consider caching for reference values***

For data that is often referred to but doesn’t change often (like configurations and reference data), consider using caching in plugins using ObjectCache in the System.Runtime.Caching namespace.

Example:

/// <summary>

/// Returns the string value for an Option key.

/// </summary>

/// <param name="xrmContext">CRM Context</param>

/// <param name="key">Option Key</param>

/// <returns>Option Value</returns>

public static string RetrieveOption(XrmServiceContext xrmContext, string key)

{

const string NULLTOKEN = "\_\_NULL\_";

ObjectCache cache = MemoryCache.Default;

string option = cache[cachePrefixStringOption + key] as string;

if (option == null)

{

if (xrmContext == null) throw new ArgumentNullException("xrmContext");

option = xrmContext.mcs\_optionSet.Where(o => o.mcs\_key == key).Select(o => o.mcs\_Value).FirstOrDefault();

if (option == null) option = NULLTOKEN;

cache.Add(cachePrefixStringOption + key, option,

DateTime.Now.AddSeconds(30));

}

return option == NULLTOKEN ? null : option;

}

#### Input/Output Parameters

***Verify the compliance of the input parameters of a CRM plug-in.***

A defensive programming strategy consists of verifying the compliance of input parameters of CRM plug-ins before you access them.

Use the following principles to verify the conformity of input parameters in a CRM plug-in:

* Confirm the existence of the request message property (which should be in the InputParameters collection) before you access it:
  + For example, the existence of the Target property for the CreateRequest request message class.
  + The request message class properties are defined at the following location: https://msdn.microsoft.com/en-us/library/gg307406(v=crm.6)
* Verify the compliance of the type of the Target property—for example:
  + The Entity type for the Target property of the CreateRequest request message class.
  + The EntityReference type for the Target property of the DeleteRequest request message class.
* Verify the compliance of the logical name of the entity—for example:
  + The Contoso\_myEntity logical name.
* Confirm the existence of the entity attribute before you access it—for example:
  + The existence of the Contoso\_myAttribute attribute on the Contoso\_myEntity entity.
* Confirm that the event that is raised is the one that you expected (Create / Update / Delete …).

***Modify the entity properties of the input parameters of a CRM plug-in during the pre-processing stage.***

In the pre-processing stage, the entity properties of the input parameters of the plug-ins can be updated through programming, to apply a validation rule or a specific calculation—for example:

* Calculate the value of the Contoso\_sum property by using the values of the Contoso\_amount1 and Contoso\_amount2 properties of the Contoso\_myEntity entity.

***Verify the compliance of the output parameters of a CRM plug-in.***

A defensive programming strategy consists of verifying the compliance of the output parameters of CRM plug-ins before you access them.

Use the following principles to verify the conformity of the output parameters of a CRM plug-in:

* Confirm the existence of the response message property in the OutputParameters collection before you access it:
  + For example, the existence of the id property of the CreateResponse response message class.
  + The properties of the response message class are defined in the following location: https://msdn.microsoft.com/en-us/library/gg307406(v=crm.6)

***Do not access the output parameters of a CRM plug-in during the pre-processing stage.***

The collection of the output parameters of Microsoft Dynamics CRM plug-ins is filled by the CRM execution pipeline only during the post-processing stage.

***Do not update the output parameters of a CRM plug-in during the post-processing stage.***

The collection of the output parameters of CRM plug-ins is the result of the execution of the CRM process and should not be updated after the process.

#### Pre-Processing and Post-Processing Images

***Restrict the attributes of pre-processing and post-processing images of a CRM plug-in.***

The CRM execution pipeline generates a SQL query during pre-processing and post-processing. This query returns the set of attributes that are specified on the images during the pre-processing and post-processing stages of CRM plug-ins.

To prevent a lack of performance for those queries, the attributes of images should be restricted to only the ones that are required by business needs. Still, use images (pre or post) instead of retrieve calls where applicable.

***Restrict the Filtering Attributes of a CRM plug-in.***

When registering CRM plug-ins, specify the Filtering Attributes in the step registration to limit the number of attributes in Dynamics CRM that trigger this plug-in. For example, when registering a plug-in that must run on the Update message for a Contact entity, specify the Filtering Attributes (for example Date of Birth) to restrict the triggering of the plug-in to only events when one of the Filtering Attributes is changed. Otherwise, any update to any attribute on the Contact would trigger the plug-in.

***Avoid the fetch of All Columns in your plug-ins.***

When fetching records in your CRM plug-in, specify which columns to retrieve. This becomes particularly important when you’re updating the records that are retrieved, to avoid triggering unnecessary update plug-ins registered for these entities. This is because every column retrieved in the fetch is updated during the update request, even when the value hasn’t actually changed.

***Limit operations that cascade to related entities.***

When using the [Update](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.update(v=crm.6).aspx) method or [UpdateRequest](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.messages.updaterequest(v=crm.6).aspx) message, avoid setting the OwnerId attribute on a record unless the owner has actually changed. When you set this attribute, the changes often cascade to related entities, which increases the time that is required for the update operation. More information: [Cascading behavior](https://msdn.microsoft.com/en-us/library/841dbfc8-05e1-4992-83ac-b9d47eab1e7b(v=crm.6)#BKMK_CascadingBehavior)

***Avoid registering plug-ins on frequently triggered messages***

For example, avoid registering your plug-in on the Execute message. Since this message is triggered frequently in Dynamics CRM, and your plug-in will run everytime this message is triggered. This will have negative implications on the performance of the Dynamics CRM server.

***Define a naming convention for pre-processing and post-processing images of a CRM plug-in.***

Applying a naming convention enables the user to standardize the handling of the images of a CRM plug-in—for example:

* Name the pre-processing image for an entity named Contoso\_myEntity as Contoso\_myEntityPreImage
* Name the post-processing image for an entity named Contoso\_myEntity as Contoso\_myEntityPostImage

***Verify the compliance of the images of a CRM plug-in.***

A defensive programming strategy consists of verifying the compliance of the images of a CRM plug-in before you access them.

Use the following principles to verify the compliance of the images of a CRM plug-in:

* Confirm the existence of pre-processing or post-processing images in the respective PreEntityImages and PostEntityImages collections before you access them—for example:
  + The existence of the Contoso\_myEntityPreImage property in the PreEntityImages collection.
* Verify the compliance of the type of the image property—for example:
  + The Entity type for the Contoso\_myEntityPreImage property in the PreEntityImages collection.
* Verify the compliance of the logical name of the entity—for example:
  + The Contoso\_myEntity logical name.
* Confirm the existence of the expected entity attribute before you access it—for example:
  + The existence of the Contoso\_myAttribute on the Contoso\_myEntity entity.

***Do not access post-processing images of a CRM plug-in during the pre-processing stage.***

The collection of post-processing images of CRM plug-ins is filled only during the post-processing stage by the Microsoft Dynamics CRM execution pipeline.

***Use pre-processing and post-processing images for read-only access.***

The collection of pre-processing and post-processing images corresponds to the CRM entities in the database. Those images should be accessed by using read-access rights only, before and after the CRM transaction that triggered the execution of the plug-ins.

#### Isolation Levels

***Take into account the design constraints in the Sandbox isolation level.***

Some restrictions apply to plug-ins registered to run in Sandbox isolation mode and must be considered when these plug-ins are designed. For a complete list for these constraint, [please see the MSDN](https://msdn.microsoft.com/en-us/library/gg334752(v=crm.6)).

***Favor the Sandbox isolation level for execution of a CRM plug-in.***

The Sandbox isolation level for execution of CRM plug-ins provides the following assets in the operating environment:

* CRM online plug-ins must run in Sandbox isolation mode
* The CRM plug-in execution process is dedicated to a specific CRM organization.
* Statistics that relate to the execution of CRM plug-ins are generated automatically.
* Execution that uses a service account is likely to have fewer access privileges than the one used in the CRM application pool.
* To extend the execution timeout while debugging (default is 30 seconds), configure this registry setting: **HKLM\SOFTWARE\Microsoft\MSCRM\SandboxDebugPlugins** (set DWORD value to 1) and attach the debugger to the **Microsoft.Crm.Sandbox.WorkerProcess.exe** service process for Sandbox isolation mode.
  + This registry setting should be configured in the base image of the developers’ virtual machines.
  + It should not be set for production environments.
  + More details: [Debug a plug-in](https://msdn.microsoft.com/en-us/library/gg328574.aspx)

#### Execution Modes

***Take into account the execution constraints that relate to the synchronous execution mode of a CRM plug-in.***

The execution constraints that relate to the synchronous execution mode of CRM plug-ins have to be taken into account.

* Take into account the execution time that is required by CRM plug-ins in a synchronous execution mode, to define the response-time objectives for the user.
* Optimize the response times of CRM plug-ins in a synchronous execution mode, to minimize their impact on the user response times.
* Size the CRM platform to minimize the impact of response times that relates to the synchronous execution of CRM plug-ins on the user response times.
* Take into account that plug-ins trigger in Web Services access used within data imports, for instance.

***Favor the synchronous execution mode for execution of a CRM plug-in.***

The synchronous execution mode for CRM plug-ins provides the following assets in the operating environment:

* It is possible to roll back the CRM operation in a transactional way that triggers the CRM plug-in.
* Immediate feedback is provided to the user to inform the user of the status (success or failure) of the CRM operation launch.

#### Processing Stages

***Use the event pipeline prevalidation stage to apply the validation and enrichment business rules in the pre-processing stage in the input parameters of Microsoft Dynamics CRM.***

Business rules that relate to input-parameter validation and enrichment can be (for example):

* Confirmation that the input parameters belong to a specific list of data.
* Calculation of some input parameters that are based on other input parameters, and pre-processing of available images.

***Do not use the event pipeline pre-operation stage to apply the business rules that relate to validation and enrichment of the input parameters of the Microsoft Dynamics CRM operation in the pre-processing stage.***

This pre-operation processing stage is completed in the Microsoft Dynamics CRM transaction that is triggered by the Microsoft Dynamics CRM operation before its execution; in terms of server resources, this is much more demanding than the prevalidation stage. Its use is limited to uncommon business rules, such as:

* The automatic generation of a parent entity to which the child entity (that triggered the Microsoft Dynamics CRM operation) should be automatically associated.

***Use the transactional post-operation processing stage to apply the business rules that relate to creation, update, or deletion of related CRM data in a transactional post-processing stage.***

This post-operation processing stage occurs in the transaction that is triggered by the CRM operation after its execution, and it enables the extension of the CRM operation in a transactional way by creating, updating, or deleting related CRM data.

However, to prevent unwanted repetitive operations, a post-update should not refer to the record that triggered the operation.

#### Types of Deployment

***Favor the database type of deployment for the deployment of a CRM plug-in.***

The database type of deployment provides the following assets:

* Automatic deployment of CRM plug-ins on front-end servers of the CRM platform
* Compatibility with the standard delivery mode of CRM solutions
* Compatibility with the online deployment of CRM plug-ins

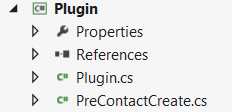
#### Plug-ins Unit Tests

It is important to test throughout the development lifecycle. Leaving the testing process to the end of a project or milestone significantly increases risk, as the quality of the application is unproven.

It is possible to adjust the default structure of CRM plugins to code unit tests with Visual Studio. Coded unit tests can be reused to avoid future regressions and give developers confidence to refactor code. Coded unit testing also tends to encourage good design principles and reduces the need for manual testing and creating test harnesses.

It is also possible to use Visual Studio to measure unit test code coverage.

By default, the Developer Toolkit scaffolds a Visual Studio project with the core components required to create a plugin.



The template hard codes the dependency on the execution context which makes isolated unit testing impossible. The context requires a connection to CRM which is not practical for unit testing.

// Construct the Local plug-in context.

LocalPluginContext localcontext = new LocalPluginContext(serviceProvider);

An approach to mitigate this is to wrap the context in a façade. An interface allows you to abstract the dependency.

public interface ILocalContext

{

void Associate(string entityName, Guid entityId, Microsoft.Xrm.Sdk.Relationship relationship, Microsoft.Xrm.Sdk.EntityReferenceCollection relatedEntities);

void Delete(string entityName, Guid id);

void Disassociate(string entityName, Guid entityId, Microsoft.Xrm.Sdk.Relationship relationship, Microsoft.Xrm.Sdk.EntityReferenceCollection relatedEntities);

Guid Execute(Microsoft.Xrm.Sdk.Entity entity);

OrganizationResponse Execute(Microsoft.Xrm.Sdk.OrganizationRequest request);

Entity Retrieve(string entityName, Guid id, Microsoft.Xrm.Sdk.Query.ColumnSet columnSet);

EntityCollection RetrieveMultiple(Microsoft.Xrm.Sdk.Query.QueryBase query);

void Update(Microsoft.Xrm.Sdk.Entity entity);

Entity TargetEntity { get; }

}

The wrapper façade would need to be implemented in a partial plugin class. This is because the original context is an internal class an only available within the scope of the plugin class.

public partial class Plugin

{

protected class LocalContext : ILocalContext

{

private LocalPluginContext \_localPluginContext;

public LocalContext(LocalPluginContext localPluginContext)

{

\_localPluginContext = localPluginContext;

}

...

The façade may now implement all the required methods using the original context (LocalPluginContext).

In the generated plugin instance class, you can now utilize the wrapper façade to interact with CRM.

protected void ExecutePreContactCreate(LocalPluginContext localContext)

{

if (localContext == null)

{

throw new ArgumentNullException("localContext");

}

ExecuteInternal(new LocalContext(localContext));

}

public void ExecuteInternal(ILocalContext localContext)

{

var entity = localContext.TargetEntity;

entity["creditlimit"] = 5000;

localContext.Update(entity);

}

A unit test can easily fake the dependency and perform isolated tests on the plugin functionality.

[TestMethod]

public void Plugin\_updates\_creditlimit\_to\_5000()

{

var plugin = new PreContactCreate();

var context = A.Fake<ILocalContext>();

// set property to return test data

Entity testEntity = new Entity();

A.CallTo(() => context.TargetEntity).Returns(testEntity);

plugin.ExecuteInternal(context);

Assert.IsTrue(testEntity.Attributes.ContainsKey("creditlimit"));

Assert.IsNotNull(testEntity["creditlimit"]);

Assert.AreEqual(5000, testEntity["creditlimit"]);

A.CallTo(() => context.Update(A<Entity>.Ignored)).MustHaveHappened();

}

### Custom Workflow Activities

***Return base.Execute(executionContext) to indicate that the activity has ended.***

At the end of the Execute method of the activity workflow, use the following command line:

return base.Execute(executionContext);

***Use the InvalidPluginExecutionException class to raise errors within specific activities.***

The InvalidPluginExecutionException error message will appear within the Microsoft Dynamics CRM workflow instance progress form (System Jobs) and will allow the user to identify why the workflow stopped.

### Ribbon

#### XML Configuration of the Ribbon

Microsoft Dynamics CRM enables the configuration of the standard application ribbon to include:

* Hiding existing buttons
* Modifying the behavior of existing buttons (for example, the ability to click the button or a specific action that is launched when this button is clicked)
* Adding new buttons
* Adding new button groups
* Adding new tabs to the existing ribbon

For each customizable entity, it is possible to modify the ribbons of:

* The entity form
* The main view of the entity (grid of all records)
* The associated view of the entity (that is, the view that exists on the parent entity of a relationship)

The ribbon is stored in Dynamics CRM in XML format, and the editing of the XML ribbon is very tedious work if it is performed manually.

Several tools are available on the web to assist developers in performing this task. Two of the most adopted tools in the community are the [CRM Visual Ribbon Editor](http://crmvisualribbonedit.codeplex.com/) and Ribbon Workbech (also available as a plugin in [XrmToolBox](http://www.xrmtoolbox.com)). It provides a graphical interface for the ribbon XML to allow developers to configure the ribbon in Dynamics CRM.

***Use a tool that is external to CRM, to modify the functionalities of the ribbon.***

Because the ribbon is very complex, it is necessary to exercise care when editing the ribbon manually. If using a community tool, verify supportability and adoption of the tool before using it.

***Modify the behavior of a standard button, instead of creating a new button.***

If you want to add a business rule to an existing button (for example, not to allow access to the Disable button for specific business roles), you should modify the validation rules of the button to add that condition.

The command that is used by the standard button will have to be customized to add the EnableRules (a call to a JavaScript function), to confirm whether the current user has a business role that gives the user access to this button.

***Develop JavaScript for the ribbon only for actions that you cannot customize within the XML of the ribbon.***

Many actions that do not require JavaScript development can be customized in the ribbon. Use of JavaScript functions should be avoided as much as possible, to decrease load time of the form or grid ribbon.

#### Extending the Ribbon using JavaScript

The behavior of the ribbon in Dynamics CRM can be extended using custom Javascript code in Web Resources. When configuring the ribbon to trigger Javascript functions in Web Resources, it is possible to pass parameters that provide contextual details of the event that triggered the function. For example:

* the number of selected elements within a grid
* the unique identifier of the current record
* or the type of the current record

For the complete list of types of parameters, you can refer either to the SDK or to the following link <https://msdn.microsoft.com/en-us/library/gg309332(v=crm.6)>

***Use a common web resource for all developments that are realized within the ribbon of an entity.***

To ensure that the ribbon features take a minimal time to load, JavaScript functions that are used within an entity should be gathered in a common web resource.

***Do not call functions that belong to another web resource.***

A customized ribbon web resource should be independent from any other web resource. In fact, the ribbon will load only the web resource that is defined in its XML configuration.

### Site Map

Like the Ribbon, Microsoft Dynamics CRM stores the definition of the application navigation in the Site Map XML.

***Define the Entity attribute or the Privilege sub-element when a new element of the site map is created.***

If you wish to have a link in the Dynamics CRM sitemap menu only appear for certain people, this can be accomplished by adding the privilege tag to the subarea in the sitemap XML. This allows you to specify that users must have the permission specified to see the link.

***Keep the Site Map structure simple***

To improve user adoption and performance of Dynamics CRM, limit the number of areas/sub-areas defined in the Site Map. Also, conditional Site Map definitions require ongoing maintenance to address changing conditions or new additional areas. It is recommended that the Site Map be standardized as much as possible to reduce this overhead.

***Modify the XML site map in Visual Studio.***

Visual Studio allows the user to modify an XML file according to the schema that is defined in an .xsd file.

The XSD schema files, which can be found in the schema directory of the SDK, should be referenced to edit the site map properly. For more information on how to use schema files to validate customization XML, please refer to the [MSDN](https://msdn.microsoft.com/en-us/library/gg334493(v=crm.6).aspx).

## CRM Online Considerations

Creating solutions for the cloud is different. It requires thinking about the system architecture in a different way. There are some nuances that must be considered carefully when designing solutions to run on Microsoft Dynamics CRM Online. Below are some of those considerations:

***Asynchronous workflows don’t run immediately.***

Asynchronous workflows are sent to the Async queue on the Async server, a shared resource. Your workflow job may have to wait its turn in the queue before it executes; although the execution time is often very quick, there’s no guarantee that it will be. If the impact of the workflow will be small enough, consider synchronous workflows.

***You can request database indexes created for your organization database.***

While you don’t have access to the database to create your own indexes, you can submit a request to Microsoft Dynamics CRM Online to have (an) index(es) created for you.

***Enforced Limits***

In Dynamics CRM Online, plug-ins have a 2 minute time-out limit, and SQL database transactions are capped at 30 second timeout. Design your plugins and data model carefully.

***Azure services region***

When using Azure services to run long running code for your Dynamics CRM Online solution, make sure your Azure service runs in the same region as your Microsoft Dynamics CRM Online organization to achieve minimal latency.

***No Access to Machine Resources***

You can’t use the web.config in CRM code. You don’t have access to the file system or the GAC.

***Only Sandbox Isolation Mode for Plug-ins***

No support for registering plug-ins to disk or GAC

***Only FetchXML Reports***

When building reports for Microsoft Dynamics CRM Online, you need to write SSRS report queries in FetchXML. But once the query is written and the data is in your possession, you can build your report using the Business Intelligence Design Studio (BIDS) just like in on-premises. Upload the RDL to Microsoft Dynamics CRM Online just like in on-premises.

***Email Router***

You can deploy the email router to Azure IaaS if you need to. However, server-side synchronization is a better alternative that simplifies CRM and email interoperability by allowing you to configure it without installing the email router component.

***Storage***

Customers only get 5 GB when they sign up for Microsoft Dynamics CRM Online, however, additional storage can be purchased. When designing your Microsoft Dynamics CRM Online solution, consider storage limitations and alternatives (like SharePoint Online, and Azure storage) for storage intensive objects.

# LINK Development Naming Amendment

## Processes

Use the following template for processes, like WFs, BPFs, actions … etc.

[<Module>][<Type>][<Entity>] <Function>

E.g. [ROC][WF][Case] Send an email to admin

For BPFs, create the process first with a ‘BPF’ prefix, and then rename according to the above convention.

## Plugins

Use the following template for plugins and custom steps.

LinkDev.<Project>.<Module>.<Stage><Function>Plugin

E.g. LinkDev.Dhca.Contact.PreGenerateAutoNumberPlugin

## Fields

* Boolean
  + Prefixed by “is”
    - E.g. ldv\_isManaged
* Lookup
  + Suffixed by “Id”
    - E.g. ldv\_ecifId
  + For fields that end with ID, add a ‘Text’ postfix
    - E.g. ldv\_nationalIdText
* OptionSet
  + Suffixed by “Code”
    - E.g. ldv\_contactTypeCode
* Currency
  + Suffixed by “Amount”
    - E.g. ldv\_creditAmount

## Relations

* 1:N
  + ldv\_[entity1]\_[entity2]\_[lookup]
* N-N
  + ldv\_[entity1]\_[entity2]

## Web Resources

* ldv\_/[project]/[module]/[filename].js
  + Functions that relate to a client form
  + E.g. ldv\_/Dhca/Contact/ContactForm.js
* ldv\_/[project]/[module]/[filename].htm
  + HTML page that is deployed on the client record
* ldv\_/[project]/[module]/[filename].png
  + PNG (graphics) file
* ldv\_/[project]/[module]/[filename].css
  + Style sheet of the HTML file

# LINK Development Developer Amendment

## Coding

### General

* Allow the system to automatically assign GUIDs
  + This allows CRM to take advantage of sequential GUIDs, which provide better SQL performance.
* Use early-bound types
  + This allows for the verification of entity and attribute names at compile-time and avoids unexpected issues.
* Use late-binding
  + Use the Entity class when your code must work on entities and attributes that aren’t known at the time the code is written.
* Limit the data you retrieve
  + When you use the methods that retrieve data from the server, retrieve the minimum amount of data that your application needs. You do this by specifying the column set, which is the set of entity attributes to retrieve.
  + For example, it is rarely a good idea to retrieve all the metadata with the RetrieveAllEntitiesRequest message, specifying the EntityFilters.All entity filter for the EntityFilters property. Instead, you might achieve better performance if you restrict the entity filter, or use one of the following messages: RetrieveEntityRequest, RetrieveOptionSetRequest, RetrieveAttributeRequest, RetrieveRelationshipRequest or RetrieveMetadataChangesRequest. The RetrieveMetadataChanges message allows for constructing a query to return just the metadata you need or the metadata that has changed.
* Limit operations that cascade to related entities
  + When you use the Update method or UpdateRequest message, do not set the OwnerId attribute on a record unless the owner has actually changed. When you set this attribute, the changes often cascade to related entities, which increases the time that is required for the update operation.
* Improve service channel allocation performance
  + Either by reusing the established service connection object, or by using:
    - IServiceManagement<IOrganizationService> orgServiceManagement = ServiceConfigurationFactory.CreateManagement<IOrganizationService>(new Uri(organizationUrl));
    - AuthenticationCredentials authCredentials = orgServiceManagement.Authenticate(credentials);
* Never access the DB directly
* You should put the CRM web service URLs into a configuration file, for example, into an app.config file, so that your code is isolated from changes to the URL. This is applicable to web-services and console/Windows applications.
* Keep a historical trail of all code and solutions (TFS)
* Use "no-lock='true'" in FetchXML when possible, and set 'NoLock' to 'true' in QueryExpressions
* Use 'Update' instead of 'SetStateRequest' in CRM 2016

### Plugins

* Write plug-ins that execute faster
  + Always write a plug-in that takes the least time to perform its intended task. For example, the Execute method is frequently processed in CRM. If you register a plug-in on that message, your plug-in can have a significant performance impact on the system because it executes every time that the Execute method is processed.
  + If you intend to register your plug-ins for synchronous execution, we recommend that you design them to complete their operation in less than 10 seconds.
* You should throw an InvalidPlugInExecutionException in your code. This error will be shown in the workflow instance form.
* Don't use threading in plugins
* Don't use 'ExecuteMultiple' or 'ExecuteTransaction' in plugins
* Don't use fields or properties in a class that inherits from the 'IPlugin' interface
* Don't use 'depth' check in plugins
* Use images instead of retrieving the entity in a plugin
* Do not associate the plug-ins on a retrieve or RetrieveMultiple request that happens every time a user views data in the system.
* Try to not update the primary entity in a plugin, instead, if possible, use pre-operation and images
* Group related plugins into a single assembly
* Extract logic that takes a long time to run into a web-service. The result should be applied to CRM asynchronously though the service itself.

### JS

* Use web resources instead of pages that require server-side processing whenever possible. If your requirements can only be achieved by using server-side processing, adhere to the requirement that your custom webpages are installed in a separate website from CRM
* If you have code that depends on another library to be fully downloaded and initialized, the most straightforward approach is to combine both libraries within a single JavaScript web resource with your code below the library code.
  + A more sophisticated approach is to use libraries such as head.js or require.js to control how the separate libraries are loaded.
  + For example, early-bound entities in JS require SDK.SOAP library to be loaded first before the browser engine could parse the file containing the entities; combining both files into one would prevent an ‘undefined’ error from popping up, which occurs when CRM loads the entities file before the library.
* Avoid loading all scripts in the Onload event
  + If you have code that only supports OnChange events for fields or the OnSave event, make sure to set the script library with the event handler for those events instead of the OnLoad event. This way loading those libraries can be deferred and increase performance when the form loads.
* We do not recommend using the addOnChange method within the OnLoad event handler simply as a matter of convenience. While this may reduce the number of steps necessary to add your event handlers, it causes the form to load more slowly.
* An IFRAME will be refreshed when the tab is expanded. Any changes to the src property will be removed. If you interact with the src property of an IFRAME, you should always include this code in the TabStateChange event instead of the Onload event.
* Avoid using sync calls in AJAX
* Use '===' instead of '=='
* Remove all 'debugger' statements
* Don't access the DOM
* Use OData v4 (Web API) instead of OData v2 or SOAP endpoint
* Use 'Xrm.Utility.alertDialog(message [, okCallback])' instead of the browser's alert
* Use 'getClientUrl' instead of hardcoding the URL in code
* Minify JS code before deploying the solution to the production server, which speeds up loading the form

## Customisations

* If possible, disable unused plug-ins
* Execution timeout
  + A plug-in registered for synchronous or asynchronous execution is restricted to complete its execution in a two-minute time limit.
  + Workflows work well for either short or long processes. However, each activity in a workflow cannot take longer than two minutes to complete.
* Limit the number of entities that are enabled for offline use
  + Carefully consider if an entity must be available for people while working offline.
  + Each entity that you enable for offline capability directly affects the time required for people to synchronize data when they come back online.
  + This is especially true for people with less powerful computers.
* Using custom entities and attributes
  + Always use the entity schema name to refer to a custom entity in code and queries. Do not use the object type code (also referred to as entity type) code because this integer value varies for custom entities in different organizations.
* Asynchronicity
  + Synchronous plug-ins can increase the platform response time because they are part of the main platform processing.
  + Asynchronous plug-ins have less impact on server response time because the code is run in a different process.
* Users can interactively create workflows in the web application, so this allows power users to edit workflows as they see fit, which increases maintainability.
* Single WF vs Many WFs
  + We can divide the WF logic into smaller WFs that are called from a single parent WF to make logic easier to maintainable.
  + This approach is slower, but it is more manageable if you frequently change your workflow definition.
  + However, CRM incurs overhead when it starts each workflow instance. The overhead occurs when all entities that are used in the workflow are retrieved and the child workflow is started in a two-step process that includes a 'Workflow Expansion Task' and the actual workflow instance.
  + Therefore, for fastest execution, use a single long workflow if possible.
* Use plug-ins to apply business logic regardless of how the data changes are made. For example, validations critical to business must be done on the server side (plugins), in addition to being done on the client side (JS, for speed).
* Avoid including unnecessary JavaScript web resource libraries
* Use collapsed tabs to defer loading web resources
  + When web resources or IFRAMES are included in sections inside a collapsed tab, they will not be loaded if the tab is collapsed. They will be loaded when the tab is expanded. When the tab state changes, the TabStateChange event occurs. Any code that is required to support web resources or IFRAMEs within collapsed tabs can use event handlers for the TabStateChange event and reduce code that might otherwise have to occur in the OnLoad event.
* If you are using tabs, place the most important attributes in the first tab on the form, and collapse tabs when possible
* Set default visibility options
  + Avoid using form scripts in the OnLoad event that hide form elements. Instead, set the default visibility options for form elements that might be hidden to not be visible by default when the form loads. Then, use scripts in the OnLoad event to show those form elements you want to display.
* If your IFRAME depends on access to the Xrm.Page object of the page or any form event handlers, you should configure the IFRAME so that it's not visible by default.
* Be sure to accurately set the Scope of your workflow before proceeding with the workflow logic. It is extremely important to understand how a workflow will run if it is a user-level process versus how the same workflow will run if it is an organization-level process!
  + The scope set in the WF itself limits the WF trigger to the scope’s owned record.
  + For example, “if the Workflow scope is ‘User’ – only the CRM records owned by the Workflow’s Owner will be triggered if it meets the Workflow’s automatic condition. If the setting is ‘Organization’ – any CRM records could trigger the Workflow if they meet the Workflow’s condition.” [3]
* Always try to cleanly end workflows. Use Stop Workflow steps wherever possible to ensure all workflow logic exits cleanly and no system resources are consumed unnecessarily.
* Infinite Loops
  + Understand the limits on recursion and how to circumvent those restrictions. For example, you may build in a 10 minute wait in a recurring workflow to ensure there are no infinite loopback issues.
* Always use good naming conventions when creating processes. While it may be clear to the person creating the workflow, it is not always clear to others what “Account Workflow #1” or “Joe CRM’s Opportunity Dialog” does! As best practice, name a process close to what the process does. (ex: “Set The Owner Based on Zip Code when Lead is Created” or “Assign New Cases to Bob Smith for Follow-Up”)
* Consider situations where you are using Wait conditions. Each Waiting workflow in the system consumes a small amount of system resources. While this may not be a concern for most implementations, this situation can cause issues in large deployments with numerous waiting workflows on hundreds or thousands of records. Use Timeout whenever possible to avoid system resource allocation issues.
* Use a specific publisher for solutions and always edit from within that solution
* Do NOT develop in your Production environment
* Use Quick Create forms wherever possible
* Use subgrids for frequent and small sized data, and associated view for larger more detailed look
* Lessen the usage of Quick Views; e.g. show the contact number of the account on a case form
* Lessen the number of stages and steps in a process flow
* Group related fields into meaningful tabs/sections
* Avoid customising a dependence on using the navigation bar
* Use role-based forms instead of role conditioning on forms using JS
* Colour same-purpose entities the same colour, or highlight the key entities only; and keep the number low
* Add meaningful description to fields
* Remove unused entities from the sitemap
* Reduce the columns in QuickFind

## Security

* Create roles according to the security best practice of least privilege, providing access to the minimum amount of business data required for the task. Assign users the appropriate role for their job.
* Use teams to create cross-functional groups so that specific objects can be shared with the team.
* At least one user must have the 'System Administrator' role at all times.
* Avoid elevation of privilege
  + Elevation of privilege attacks occur when a user can assume the privileges of a trusted account, such as an owner or administrator. Always run under least-privileged user accounts and assign only needed permissions. Avoid using administrative or owner accounts for executing code. This limits the amount of damage that can occur if an attack succeeds. When performing tasks that require additional permissions, use procedure signing or impersonation only for the duration of the task.
* For improved security, make sure that your separate website runs on a different account from CRM. This account should have the minimum access possible and one that does not have direct access to the Microsoft databases. You can use a complex password that doesn’t expire because no person signs in to this account – only in to your application.
* Avoid use of ActiveX controls because they have known security problems.
* Always use a confirmation dialog box when you delete records or apply sensitive changes
  + For example, when the business process reaches the ‘payment’ stage, we could provide a confirmation message when the user clicks on ‘initiate payment’.
* Don’t use anonymous access.
* Use integrated Windows authentication, NTLM, or Basic authentication over Transport Layer Security (TLS) or Secure Sockets Layer (SSL).
* Use TLS/SSL to avoid sending unencrypted data over the network if your website is on a different computer than CRM.
* Use the Restrict cross-frame scripting, where supported option when you don’t fully trust the content displayed in an IFRAME.
  + Browser plug-ins are disabled.
  + Forms and scripts are disabled.
  + Links to other browsing contexts are disabled.
  + Content is treated as from a different domain even if the domain is the same.
* Use teams instead of sharing

## Tools

* ‘Ctrl+Shift+Q’ in any form to measure form load performance

# LINK Development UX Amendment

## Design Target

### Design for the specialist role

Most implementations have specialist and generalist roles. Good UX design is seldom generic.

Design experiences optimised for each specialist role

Optimising experiences for the generalist role makes it hard for the specialists to use the system. It is important to check for this pitfall right after the solution design phase.

### Design for the business user

Design of the user experience is targeted at the end business user following a business process. It is important to weigh each decision from the end user’s point of view.

One way to mitigate risks is to make low cost prototypes and test them on real business users before finalizing designs. This can help discover usability issues early.

For example, CRM implementation aligned to management than it is aligned to end user needs. Such an objective leads to poor user perception, poor user adoption, and low business impact because users use it only at the minimum they need to in order to satisfy management.

## Dashboards

### Purpose

Dashboards should be the home page from where 80% to 90% of day-to-day tasks for most roles should start. Ideally, all or most business flows should start from the dashboard. It should act as that initial gateway into a user’s primary work.

Role-based dashboards ideally should be designed for each role, answering key questions. Consider specific roles individually and provide the information and insight vital to their role, rather than have a generic dashboard for all.

### Guiding questions

|  |  |
| --- | --- |
| Question | Visualization |
| What do I need to do to understand the state of business and my work? | Chart of important trends |
| What are the top tasks for the day? | Queue of activity I need to pick up, My Activities and tasks |
| Whom do I care about? | View of key accounts and contacts |
| How am I doing? | Performance |

## Navigation Bar

### Purpose

The navigation bar is not intended to be a part of a contextual business flow. It takes multiple clicks and visual searching to find the right area or entity to get to. These tasks make sense for data set exploration, for example, if the user is looking for a set of records via a view for an entity less commonly used.

### Design

* Simplify the navigation as much as possible by hiding (or disabling) menus.
  1. Open the XrmToolbox and modify the sitemap
  2. Remove related records in the navigation, which have already a sub-grid on the form
  3. Put the most used entities first
  4. “Is there another place where users see this info and is this logical?”
     + Yes: remove
     + No: keep
* Make each process area self-sustained
* Configure a landing page for each area – dashboard

## Forms

### Design and purpose

* Use role-based forms instead of role conditioning on forms using JS.
* Rename forms to reflect their purpose.

### Layout

“Millers Law” is a valuable principle to understand and follow: Humans can process approximately seven “chunks” of information at a time.

This can be done by recognizing and representing natural structures and collections of related data together in the user interface. Breaking larger concepts, like an account, down to meaningful parts such as the company details, financial details, and contact information can assist with this. Each of these can then be represented either on a form or as separate entities.

Rename tabs and sections to reflect their purpose, or hide the label if unnecessary or obvious, and group related fields into meaningful tabs/sections.

### Fields

* Make sure field labels are not cut off if too long.
  + Increase the size of the field label from the properties dialogue of the field.
* Dynamics CRM comes with a lot of standard fields, right out of the box, which are useless to most of users.
  + Remove fields which are not used
  + Use business rules to hide/show fields
  + Use custom code to hide/show entire tabs
* Nobody likes to put in dozens of hours, inputting data into the system. This should be automated and simplified as much as possible.
  + Make sure users never need to input data two times
  + Use business rules to set field values based on other fields
  + Use workflows to create records and data
  + Use Business Process Flows to guide users in the process
* Show creation and modification timestamps and related users.
  + Add the ‘created/modified on’ and ‘created/modified by’ fields
* Identify events that should raise a flag and create a post or note, or update a field.
* Avoid redundant field naming; e.g. ‘case title’ vs ‘title’.
* Group fields with short prefixes to aid in advanced find.
* Add a meaningful description for fields, which will pop up to the user when he hovers over the field label on the form.

### Quick create forms

It is important to only have minimum fields that would be always readily available at creation. The user can transition to the full form for further updates or edits if needed. Quick create forms also allow the main form to be designed and optimized for the consumption experience.

Quick create forms can be used in two scenarios:

* To quickly create related information without losing flow. The form is launched from the sub-grids on a form.
* To quickly create information in a new unrelated context. The form is launched from the Navigation Bar.

## Views

* Make as much data available on one screen, without overcrowding it. Users like to have as much info as possible, without needing to click through to another record.
* Modify views so that they contain all the important fields.
* Adjust the sorting of your records and the width of the columns
* Consider sorting by ‘created on’ or ‘modified on’, or create a dedicated view for it.
* Synchronise column order and width across views as much as possible.
* Don’t use look up fields in the first column
  + Usually the user will click on the first column cell to access a record, and this redirects to an unintended record.
* Remove default views that are not used.

### Quick view forms

Quick view forms were created to provide a quick preview of key attributes of a related entity without unnecessary navigation to the related record. This works by surfacing key information on a parent entity form through configuration.

They were not intended for building complete de-normalized forms where multiple fields from multiple related entities are all shown in the parent form.

Quick view should be used sparingly:

* To show key information only, not replicate the entire record.
* To show information directly actionable or related to the outcome driven from the parent record.

For example, showing the email and phone number of a primary contact on an account form is an ideal use case for quick view forms where the likely action is to realize you want to contact the primary contact for the account.

## Grids

Sub-grids and lookups are used to provide a natural flow for quick in-context information discovery and consumption.

### Sub-grids

Use sub-grids on a form, when the related entity information in question is needed frequently and the number of records that need to be visible for effective consumption is low (<10). It is important to ensure that information that’s key to the decision making process, like owner, is visible on the sub-grids.

For example, if there are, on average, three to five opportunities per account and the key fields needed in the context of an account are only Opportunity Name, Owner, Estimate Revenue, and Estimated Close Date, it is ideal to show the opportunities associated with an account as a sub-grids on the opportunity form. If it was necessary to know what the status of all the opportunities were to determine if they are relevant to the current process, it would be important to add that to the sub-grids as well. This avoids the frustration of users needing to open up each opportunity record individually to check the status – a simple change that has huge user experience benefits.

### Associated grids

Associated grids are used to provide a more detailed immersive experience for related entity information. Use associated grids on a form when the related entity information in question is needed infrequently or if the number of records that need to be visible for effective consumption is moderate to high (>10). Associated grids take more effort and clicks to get to. However, they provide a dedicated experience to explore and consume a large number of records.

An ideal example is the out-of-the-box (OOB) associated grid for activities on the account form. A large account might have hundreds of activities regarding it. However, it is of interest only in a few situations like escalations or while analysing the loss of a deal. Hence, it makes perfect sense to show the activities associated with an account as an associated grid.

Remove menu items that already have a sub-grid on the form.

Prioritise the order of associated grids based on access frequency.

### Editable grids

One of the core reasons to look at editable grids is the ability to streamline the user interface for users. They no longer need to access multiple screens to view and edit records, and combining this with Quick Create forms, you no longer need to leave the screen.

Configuring editable grids is quite straightforward. Editable grids can be configured at the entity level or from the form

1. Entity level
   * Access the entity in your solution and open the Controls tab. Here you can add the Editable Grid control and determine which client has access. Now all views are editable.
2. Form level
   * Add sub grid to the form, and access the controls tab on the properties.

## Business Process Flow

Another key purpose of the process bar is to link together related entities that are tied to a process through automatic form transitions.

Use the business process to trim the form by reducing the number of available fields to only the ones related to the current stage.

## Performance

One of the major reasons for poor performance is poor scripting habits.

### Avoid scripting

Try to avoid complex client-side scripting. Look at alternative options like server-side code or design change. For example, instead of showing and hiding some fields based on two roles, it might be better to create two roles and use two role-based forms.

### OnLoad event

Even if you choose to write client-side code, avoid functions with excessive wait times in the OnLoad event. These delays directly add to the form load time. The higher the form load time, the lower the user is perceived application performance.

### Server call-back functions

Server call-back functions in client-side code cause the most delay. If a synchronous call is made and the client is, waiting for a response from the server – to the user the client looks like it has stopped responding. Although the client is just waiting, the perception the user gets is that CRM is prone to hanging randomly.

### Plugins

Use synchronous plugins to show data changes that should always be kept up-to-date.

Use asynchronous plugins if the changes will not be reflected on the form (e.g. notifications), to reduce wait time when saving data.

## Theming

The primary purpose behind introducing theming is to help brand the CRM system along the same lines of other corporate line-of-business applications a customer might be running. This does have a tangible effect on user ownership and adoption of the system as well as enhancing the experience when the system is deployed in a customer-facing scenario.

Theming allows for branding the application with a customer icon and changing accent colours for hover and selection of certain areas and entities.

Keep the following best practices in mind while you use the theming functionality:

* Accessibility
  + Be aware of the colour contrast for new custom themes. Contrast ratio is an important measure of accessibility. Our OOB theme has the correct contrast ratios to ensure optimal usability. High contrast mode always uses the Dynamics CRM default colour settings.
* Don’t overuse colours
  + Keep the number of colour groups low. If you use too many colours, the colours lose their significance and do not add to the user experience.
* Use one of the following patterns to colour entities:
  + Make most entities a neutral colour and highlight only the key ones.
  + Make entities the same colour when the entities have a similar purpose.
* Add an icon for all entities

# Appendix: Code Reviews

Code reviews should be considered whenever a new, major change or version of a production piece of software is ready to be rolled out. It is recommended that a code review be conducted between the alpha and beta stages to catch more architectural issues, and between the beta and production stages to catch crash, leak, performance and other non-architectural issues.

## Code Review Entry Criteria

Code compiles without error or warnings where the compiler has maximum option of /W2.

Code listing available via hard or soft copy.

Line numbers are available if listing by hardcopy.

Appropriate time has been allotted for the review process.

Microsoft’s recommended best practices applied where appropriate.

Detailed specifications of the application are available.

After the review has been completed, the following exit criteria must be satisfied before the code review process is complete.

## Code Review Exit Criteria

All major defects (and any minor defects that are deems as must fixes) found by review are documented and suggested fixes provided if possible.

All minor defects that aren’t deemed as must fix are documented and suggested fixes provided if possible.

Your code should pass every design rule contained in:

* Writing Solid Code
* Code Complete

## Review Suggestions

Here are some of the things to look out for during a code review:

### Code structure

How are error conditions trapped and handled?

Are objects destroyed/released when not needed (.Net changes this a bit)?

Is there any non-required functionality or never hit code paths?

Are boundaries being checked?

Are there tight loops that could end up never terminating?

Is there thread synchronization objects being used for multi-threaded applications?

### Change Control

If change control is being documented in the header of the functions, look at what has changed recently and start there.

What type of source control process are they utilizing?

### Performance/Optimization

File I/O has been optimized for efficiency? Have the file operations been implemented so the file is opened and traversed only once to process (unless a random-access file)? (ie. watch kernel transitions)

Have database operations been implemented for efficiency? Has the correct cursor type been specified? Are indexes being used instead of traversing the entire record set? Is the correct record set returned, or are extra records included in the record set that are never used? (\*reliant on the schema of the DB’s being provided)

Have all loops been checked for optimization?

Have all recursive routines been checked for optimization and limits?

### Exception Handling

Is error handling implemented so that no unhandled exceptions can crash the program? Make sure non-terminating handlers free resources and release synchronization objects

Do structured error handlers catch all possible error types that can be thrown?

Have transactions been used to group related executions and protect against failures? Are Commit and Rollback statements implemented properly to manage transactions in case of failures? (\*Specific to DB code)

Sometimes too much exception handling can cause slow performance; i.e., don’t throw returns, and so on.

### Variables

Are there global variables (note: Global variables are initialized to NULL automatically)

Are all object variables destroyed properly? If COM, check for weak references and ensure release function takes into account.

Are data comparisons always done between variables of the same type?

Have global or local variables been initialized to default values correctly?

### Execution Control

Is “goto” only used for error handling, or a single-point of exit (instead of break)?

Do all recursive functions have limits to ensure that they will always return and never repeat until stack overflow?

In multiple if-then-else situations, are all conditions treated properly? Always recheck ternary operations (var? true:false)

## Suggested Questions

### Maintenance

Is the sophistication of this interface appropriate for its audience?

Is this code documented? Is the documentation complete and correct?

Is this code commented? Are the comments complete and correct?

Does this code follow your team’s style conventions?

Is this code readable? Could it be made more readable?

Is this code compliant with the most recent language standard? Does it use any MS extensions? Does it use any deprecated, obsolete, or undefined features?

### Coupling

Minimal and complete interfaces

Is the public interface to this module minimal and complete? Is the implementation encapsulated? Could it be more encapsulated?

Does this module have any public properties? Are they necessary? Is this class more than a glorified structure?

Dependencies

Does this module reference other modules? What mechanism is used (global variable, inheritance, containment, aggregation, function argument, and so on.)? Could a weaker mechanism be used?

What files does this file depend on? What are the compiler dependencies? What are the linker dependencies? Can you use forward declarations, abstract classes or abstract factories to reduce the number of file dependencies?

Do inheritance hierarchies have an abstract class at the root?

Do object references point to abstract interfaces? In class members? In arguments? In return types?

Does this class use multiple inheritances? Multiple subtype inheritance?

### Cohesion

Does this compilation unit (.h/.c) describe only one class?

Does this class describe a single, clear abstraction?

Does this subroutine perform a single, well-defined operation; that is, does it produce any side-effects? Are you sure?

How long is this subroutine’s argument list? Is it a class member? Are some of the arguments really parameters? If so, could they be made properties of the class instead?

Does this subroutine have a single point of return?

Does this function have multiple output values? Are they logically related to each other? Could they be encapsulated?

### Reuse

Does this system use any known design patterns?

Did you try to find a pre-existing component from a reliable source before building your own?

Does this module use any custom containers (even arrays)? Is there a suitable STL container?

Does this component export a COM interface? What is its external client?

Does this component use MFC? Is it a document editor?

Does this code anticipate a hypothetical future need? How did you determine this need? Does this feature make the code more complex?

### Performance

How big is this program’s working set? How big is it when it loads? What is the maximum size? What causes it to grow?

How fast does this program respond to user input? Is it fast enough? Is it faster than necessary? Is it fast at the expense of memory efficiency?

How many processes/threads does this program use? Could it use fewer? Does it need more? Does this program use Alertable I/O? Does it use I/O Completion Ports?

Does this program run in the background? Is it a service?

Does this program communicate with other processes? Remote processes? How does it marshal data? Is this communication asynchronous? What happens if the other process is unresponsive?

Are this program’s routines located physically near the data on which they operate, that is, does it have good locality of reference? Did you run BBT?

Does this code load resources on demand?

Is this code optimized? How? Did you measure performance with a profiler first? When was the last time you measured? What did you find?

### Correctness/Reliability

Do you know exactly how this routine/class/component works? Is it documented? Is it diagrammed in UML?

Does this module reference an external component? Is the behavior of that component completely documented?

Do you validate all subroutine/method arguments? Do you validate preconditions and post conditions? Do you validate class and loop invariants? Do you validate intermediate results with ASSERTs?

Is this Service/Driver secure in a multi-user environment? How do you know?

Does this code handle all errors? Does it use exceptions? Does it reference an external component that uses exceptions?

Are all procedures ‘exception-safe’, that is, are they transactional under error conditions?

Is this code type-safe? Have you gone to reasonable lengths to avoid type casting?

Is this interface const-correct?

If code is expected to be re-entrant, does this code acquire/release synchronization objects in a systematic way? Is there any risk of deadlock? Can you prove it?

Does this module have a unit test? Is it up-to-date? Do you run it before checking in code changes?

Does this program build without warnings?

Does this program pass Lint without warnings?

Does this program run under Bounds Checker (or other memory alloc profiler tools) without warnings?

Does this device driver pass the Driver Verifier? Does it satisfy other logo requirements?

# References

| **Reference** | **Link** |
| --- | --- |
| Microsoft Dynamics CRM Implementation Guide for CRM Online and CRM 2016 (on-premises) | <https://www.microsoft.com/en-us/download/details.aspx?id=50039> |
| Microsoft Dynamics CRM Implementation Guide for CRM 2015 | <https://www.microsoft.com/en-ca/download/details.aspx?id=45022> |
| Microsoft Dynamics CRM Software Development Kit (SDK) for CRM Online and on-premises CRM 2016 | https://www.microsoft.com/en-us/download/details.aspx?id=50032 |
| Microsoft Dynamics CRM 2015 Software Development Kit (SDK) | <https://www.microsoft.com/en-us/download/details.aspx?id=44567> |
| Microsoft Dynamics CRM 2015 and Microsoft Dynamics CRM 2016 Performance and Scalability Documentation | https://www.microsoft.com/en-us/download/details.aspx?id=45905 |
| Optimizing and maintaining the performance of a Microsoft Dynamics CRM 2011 server infrastructure | <https://www.microsoft.com/en-us/download/details.aspx?id=27139> |
| Optimizing and maintaining client performance for Microsoft Dynamics CRM 2011 and CRM Online | <https://www.microsoft.com/en-us/download/details.aspx?id=23261> |
| Deploying Microsoft Dynamics CRM 2011 and CRM Online Solutions from Development through Test and Production Environments | <https://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=27824> |
| Patterns and Principles for CRM Online Solution Builders | <https://blogs.msdn.microsoft.com/crm/2015/04/29/microsoft-dynamics-crm-online-patterns-principles-for-solution-builders/> |
| Microsoft Dynamics Sure Step | <https://mbs.microsoft.com/customersource/Global/SureStep> |
| Dynamics CRM Community | <http://www.microsoft.com/dynamics/crm/community/default.mspx> |
| Dynamics CRM on MSDN | <http://msdn2.microsoft.com/en-us/dynamics/crm/default.aspx> |
| Premier Field Engineering CRM in the Field blog | <http://blogs.msdn.com/b/crminthefield/> |
| Dynamics CRM Product Team Blog | <http://blogs.msdn.com/b/crm/> |
| Microsoft Dynamics CRM Online patterns & principles for solution builders white paper | <http://go.microsoft.com/fwlink/p/?LinkID=533946> |
| Best practices for developing with Microsoft Dynamics CRM | <https://msdn.microsoft.com/en-us/library/gg509027.aspx> |
| Dynamics CRM and User Experience (UX) | <https://crmgiant.eu/research-and-discussion/dynamics-crm-and-user-experience-ux/> |
| 10 Tips for Designing a Great User Experience in Dynamics CRM | <http://survivingcrm.com/2014/09/10-tips-designing-great-user-experience-dynamics-crm/> |